João Pedro da Silva Matos

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Uma abordagem para integração de Sistemas de Manufatura num contexto de Indústria 4.0

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: José António Barata de Oliveira, Professor Doutor, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Co-orientador: Ricardo Alexandre Fernandes da Silva Peres, Mestre, Uninova/CTS

Júri

Presidente: Doutor João Francisco Alves Martins
Arguente: Doutor Ricardo Luís Rosa Jardim Gonçalves
Vogal: Doutor José António Barata de Oliveira

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2017

Uma abordagem para integração de Sistemas de Manufatura num contexto de Indústria 4.0

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Para a minha família e namorada.

Agradecimentos

Quero agradecer a todos os que me ajudaram e apoiaram durante todo o curso, assim como durante o processo de desenvolvimento e escrita desta tese.

Em primeiro lugar, agradeço ao meu orientador Professor Doutor José Barata, pela possibilidade de trabalhar e desenvolver a minha tese no projeto H2020 PERFoRM durante o último ano. Esta oportunidades ajudou-me a crescer, quer a nível profissional quer a nível pessoal.

Ao meu co-orientador e amigo, Ricardo Peres, por toda a amizade, apoio e dedicação no desenvolvimento deste projeto. Agradeço igualmente, a partilha de conhecimento ao longo de todo o meu percurso académico.

Ao André Rocha, pela oportunidade, disponibilidade e ajuda que demonstrou durante todo o tempo em que trabalhamos juntos.

Ao meu amigo, Miguel Raposo pelo apoio e companheirismo durante todo o decorrer do curso. Sem a sua dedicação e amizade ao longo dos últimos anos, teria sido bastante difícil.

À minha namorada, Sandra Neves, um agradecimento do fundo do coração pelo seu apoio incodicional em todos os momentos. Sem o seu carinho, amor e amizade não seria possível ser quem sou hoje e chegar onde cheguei.

Por fim e não menos importante, agradeço à minha familia, particularmente aos meus pais e irmã por estarem sempre ao meu lado para tudo o que preciso.

Resumo

Nos últimos anos a mentalidade dos consumidores mudou, sendo cada vez mais exigido produtos customizados. Com estas exigências as empresas foram obrigadas a abdicar de sistemas de produção em massa para sistemas mais flexíveis, inteligentes e dinâmicos.

Com o intuito de responder a estas novas exigências, surgiram novos paradigmas e novas tecnologias e assim a Indústria está a caminhar para uma nova revolução. O nome atribuído a esta revolução foi Indústria 4.0 e foca-se em sistemas descentralizados, inteligentes, reconfiguráveis e flexíveis.

Um princípio essencial na Indústria 4.0 é a interoperabilidade dos elementos heterogéneos do sistema. Além de mais importante, este princípio é também o mais difícil de garantir, pois existem já imensos dispositivos e softwares com linguagens de comunicação e protocolos definidos e estes são muitas vezes diferentes.

Uma solução para garantir a interoperabilidade de todos os elementos heterogéneos num sistema de uma forma *standard* e genérica, para permitir a escabilidade do mesmo, é a definição e utilização de um Modelo de Dados e de Interfaces Comuns.

O trabalho desenvolvido está integrado no âmbito do projeto *Horizon 2020 Production harmonizEd Reconfiguration of Flexible Robots and Machinery* (H2020 PERFoRM), onde é proposta uma arquitetura que oferece uma infraestrutura e metodologia para implementar a nova geração de sistemas de automação na forma de sistemas distribuídos de automação baseados em cloud, seguindo todos os princípios definidos na Indústria 4.0.

O trabalho descrito neste documento contribui para o H2020 PERFoRM com a implementação do Modelo de Dados e Interfaces Comuns, com base nos vários requisitos presentes no projeto, que vai permitir uma troca de informação de forma standard e genérica.

É ainda proposta uma implementação preliminar da abordagem geral, de forma a validar o trabalho desenvolvido num ambiente simulado.

Palavras-chave: *Indústria 4.0*, *Modelo de Dados*, *Interfaces Standards*, *Interoperabilidade*, *AutomationML*.

Abstract

In the last few years, the consumer's minds has changed, showing an increasing demand for highly customized products. With such demands the companies were forced to switch from mass production systems to more flexible, intelligent and dynamic systems.

With such changes, new paradigms and new technologies have emerged and the Industry itself is on the way to a new revolution. The assigned name of this revolution is Industry 4.0, which focuses on decentralized, intelligent, reconfigurable and flexible systems.

The interoperability of every element in a system is the most essential principle in Industry 4.0. Besides being the most important, the interoperability is also the most difficult to ensure, as there are many devices and softwares with very different communication languages and protocols already defined.

One possible solution to ensure the interoperability of all elements presented in a system in a standard and generic way, ensuring system scalability is the definition and usage of a Data Model and Common Interfaces.

The work developed is part of the Horizon 2020 Production harmonizEd Reconfiguration of flexible robots and machines (H2020 PERFoRM) project, which proposes an architecture and methodology to implement the new generation of Manufacturing Systems, in form of automation cloud systems, following the principles defined in Industry 4.0.

The work described in this document, contributes to the H2020 PERFoRM with an implementation of the Data Model and Common Interfaces, based on the various project requirements, which allow a standard and generic exchange of information.

It is also proposed a preliminary implementation of the general approach, in order to validate the developed work in a simulated environment.

Keywords: *Industry 4.0*, *Data Model*, *Standard Interfaces*, *Interoperability*, *AutomationML*

Índice

Lista de Figuras **xv**

Lista de Tabelas **xvii**

Siglas **xix**

1 Introdução **1**
1.1 Descrição do Problema 1
1.2 Pergunta de Investigação e Hipóteses 2
1.3 Visão Geral do Trabalho Realizado 2
1.4 Principais Contribuições 3

2 Estado de Arte **5**
2.1 Indústria 4.0 6
2.1.1 Componentes da Indústria 4.0 7
2.1.2 Princípios base da Indústria 4.0 10
2.2 Modelação e Representação de dados 12
2.3 Middleware 14
2.3.1 Soluções Existentes 15
2.4 Conclusões Gerais 16

3 Arquitetura do PERFoRM **17**
3.1 Visão Geral do PERFoRM 17
3.2 Arquitetura do PERFoRM 18
3.2.1 Middleware Industrial 20
3.2.2 Interfaces Standard 20
3.2.3 Adaptadores Tecnológicos 22

4 Modelo de Dados Comum **25**
4.1 Modelo de Dados Comum 25
4.1.1 PMLParameter e PMLValue 27
4.1.2 PMLEntity 28
4.1.3 PMLComponent e PMLSubsystem 29
4.1.4 PMLSkill 30

4.1.5 PMLAtomicSkill e PMLComplexSkill 31
4.1.6 PMLConfiguration 32
4.1.7 PMLProduct 33
4.1.8 PMLConnector 34
4.1.9 PMLEvent 35
4.1.10 Tipos de Eventos 36
4.1.11 PMLContext 37
4.1.12 PMLSystem 37
4.1.13 PMLSimulationResult 38
4.1.14 PMLSchedule e PMLOperation 39

5 Implementação **41**
5.1 Modelo de Dados 42
5.1.1 AutomationML 42
5.1.2 Modelo de Dados 46
5.2 Interfaces genéricas e standard 49
5.2.1 Interface para o alto nível 50
5.2.2 Interface para o baixo nível 51
5.2.3 Interface para o Modelo de Dados 52
5.3 AML Parser 53
5.3.1 Modelo de Dados 54
5.3.2 Leitura de ficheiro aml 55
5.3.3 Escrita de ficheiro aml 57
5.4 Middleware 59
5.4.1 Serviços 59
5.4.2 WinCC Open Architeture 62

6 Resultados e Validação **65**
6.1 Ambiente Simulado 66
6.2 WinCC OA HMI 67
6.3 Interação entre WinCC OA e o V-REP 69
6.4 Interação com a ferramenta de Análise de Dados 73
6.5 Interação com o Escalonador 76

7 Conclusões e Trabalho Futuro **79**
7.1 Conclusões 79
7.2 Trabalho Futuro 80

Bibliografia **81**

Lista de Figuras

2.1 História das Revoluções Industriais 6
2.2 Sistema sem Middleware e com Middleware como camada de integração . . 14

3.1 Visão geral da arquitetura do PERFoRM 19
3.2 Camadas da Arquitetura do PERFoRM 21
3.3 Tipos de Adaptadores Tecnológicos 22

4.1 Modelo de Dados Comum 26
4.2 PMLParameter e PMLValue 27
4.3 PMLEntity 28
4.4 PMLEntity, PMLComponent e PMLSubsystem 29
4.5 PMLSkill 30
4.6 PMLSkill, PMLAtomicSkill e PMLComplexSkill 31
4.7 PMLConfiguration 32
4.8 PMLProduct 33
4.9 PMLConnector 34
4.10 PMLEvent 35
4.11 Tipos de Eventos 36
4.12 PMLContext 37
4.13 PMLSubsystem e PMLSystem 37
4.14 PMLSimulationResult 38
4.15 PMLSchedule e PMLOperation 39

5.1 Constituição do AutomationML 43
5.2 Formas de Modelação no AutomationML 44
5.3 Editor do AutomationML 45
5.4 Role Classes definidas 46
5.5 System Unit Classes definidas 46
5.6 PMLEntity no editor de AutomationML 47
5.7 PMLValue no editor de AutomationML 48
5.8 PMLProduct no editor de AutomationML 49
5.9 Interface de ligação com o alto nível 50
5.10 Interface para ligação com o baixo nível 51

5.11 Interface para ligação ao Modelo de Dados 52
5.12 Classes extraídas do esquema XML do CAEX 54
5.13 Classes Java do Modelo de Dados 55
5.14 Variáveis existentes na classe AMLParser 56
5.15 Variáveis existentes na classe AMLWriter 57
5.16 Métodos construtores para a classe AMLWriter 58
5.17 Resultado de um serviço em JSON 60
5.18 Interação entre um Aplicação Genérica, o Middleware e o Modelo de Dados . 62

6.1 Ambiente Simulado em V-REP 66
6.2 Ambiente Simulado em 2D 67
6.3 *DataPoint* e respetivos Nós 68
6.4 Lista de *TAG's* criadas no KepServer 70
6.5 Configuração do Servidor no WinCC OA 70
6.6 Ficheiro de configuração do projeto do HMI 71
6.7 Lista de *Managers* presentes no projeto do HMI 71
6.8 Configuração do Valor Periférico de um Nó 72
6.9 Arquitetura completa mais tecnologias 73
6.10 Fluxo de Dados desde o WinCC OA 74
6.11 Criação de Agentes segundo o ficheiro aml 75
6.12 Resultados da Ferramenta de Análise de Dados 75
6.13 Interface Gráfica do Escalonador 76
6.14 Resposta do Middleware ao Escalonador 77
6.15 Resposta do Escalonador ao Middleware 77

Lista de Tabelas

2.1 Princípios chave presentes numa abordagem para a Indústria 4.0 10

Siglas

AML	*Automation Markup Language*.
BatMAS	*Batch Process Automation with an Ontology-driven Multi-Agent System*.
BMS	Sistemas Biónicos da Manufatura.
CAEX	*Computer Aided Engineering Exchange*.
CMA	*Component Monitoring Agent*.
COLLADA	*COLLAborative Design Activity*.
CPS	Sistemas Ciber-Físicos.
DA	*Deployment Agent*.
DPWS	*Device Profile for Web Services*.
EPS	Sistemas Evolutivos de Produção.
ERP	*Enterprise Resource Planning*.
FMS	Sistemas Flexíveis da Manufatura.
FP7	*7th Framework Programme*.
H2020	*Horizon 2020*.
HMI	*Human Machine Interfaces*.
HMS	Sistemas Holónicos da Manufatura.
HSEL	*University of Applied Sciences Hochschule Emden/Leer*.
HTTP	*Hypertext Transfer Protocol*.
I-FEVS	*Interactive Fully Electrical Vehicles*.
IIB	*IBM Integration Bus*.
IoS	Internet dos Serviços.
IoT	Internet das Coisas.

IPB	Instituto Politécnico de Bragança.
JAXB	*Java Architecture for XML Binding*.
JINI	*Java Intelligent Network Infrastructure*.
JSON	*JavaScript Object Notation*.
KBF	*Key Business Factor*.
KPI	*Key Perfomance Indicator*.
LOC	Loccioni.
MAS	*Multi-Agent System*.
MES	*Manufacturing Execution System*.
MII	*Manufacturing Integration and Intelligence*.
OPC DA	*Open Platform Communications Data Access*.
OPC UA	*Open Platform Communications Unified Architeture*.
PERFoRM	*Production harmonizEd Reconfiguration of Flexible Robots and Machinery*.
PLC	Controladores Lógicos Programáveis.
PRIME	*Plug and produce intelligent multi-agent environment based on standard technology*.
REST	*Representational State Transfer*.
RICS	*Robotics & Industrial Complex Systems*.
RMS	Sistemas Reconfiguráveis da Manufatura.
SCADA	*Supervisory Control And Data Acquisition*.
SMA	*Subsystem Monitoring Agent*.
SOA	Arquitetura Orientada a Serviços.
SOAP	*Simple Object Access Protocol*.
SQL	*Structured Query Language*.
TCP/IP	*Transmission Control Protocol/Internet Protocol*.
UPnP	*Universal Plug and Play*.

WINCC OA	*WinCC Open Architecture*.
WS	*Web Services*.
WSDL	*Web Services Desciption Language*.
XML	*eXtensible Markup Language*.
XSD	*XML Schema Language*.

Capítulo 1

Introdução

1.1 Descrição do Problema

Nas últimas décadas verificou-se alterações no pensamento dos consumidores, sendo cada vez mais exigidos produtos altamente customizados. Além disso, os fornecedores são cada vez mais competitivos e possuem um leque de produtos diferenciados bastante grande.

Por todos estes motivos, muitos fabricantes na indústria adaptam cada vez mais os sistemas tradicionais de produção em massa para sistemas de produção mais inteligentes, flexíveis e dinâmicos.

Novos paradigmas e conceitos surgiram na literatura de forma a tentar combater estas exigências dos consumidores e a tornar os Sistemas de Manufatura mais adequados à atualidade. Os focos destes paradigmas e conceitos são vários, como por exemplo, reconfiguração, flexibilidade ou modularidade.

Aliando estes paradigmas, conceitos e as novas tecnologias que surgiram para os suportar vê-se que a indústria está a caminhar em direção a uma nova revolução. Esta é intitulada de Indústria 4.0.

Um princípio essencial na Indústria 4.0 é a interoperabilidade entre todos os elementos presentes no sistema. De forma a existir uma comunicação standard entre todos estes elementos, surgiu uma necessidade de encontrar uma forma de integração de componentes heterogéneos num Sistema de Produção.

Esta necessidade é sentida nos sistemas atuais presentes na indústria, pois existem muitos elementos com linguagens de comunicação próprias já definidas e as comunicações entre estes diferentes elementos torna-se bastante complexa, sem qualquer tipo de linguagem comum ao longo do sistema.

Assim sendo, tem-se uma oportunidade de investigação e na próxima secção é apresentada a pergunta de investigação.

1.2 Pergunta de Investigação e Hipóteses

Considerando as dificuldades apresentadas na secção anterior, torna-se evidente que um Modelo de Dados Comum e Interfaces Standard são essenciais nos sistemas atuais na indústria. Assim sendo, considerou-se a seguinte pergunta de investigação:

- Que descrição de um Sistema de Produção poderá possibilitar a integração transparente de elementos heterogéneos no contexto da Indústria 4.0?

Considerando a pergunta de investigação, foi proposta a seguinte hipótese:

- No contexto da Indústria 4.0 é possível a integração de elementos heterogéneos caso estes sejam descritos segundo um Modelo de Dados Comum genérico, e expondo as suas funcionalidades através de Interfaces Standard.

Na secção 1.3 vai ser apresentada uma descrição geral do trabalho desenvolvido.

1.3 Visão Geral do Trabalho Realizado

Formulada a questão de investigação, é proposto o desenho de um Modelo de Dados Comum genérico capaz de abstrair todos os elementos presentes num Sistema de Produção e a definição de Interfaces Standard para expor as funcionalidades dos mesmos.

Como o Modelo de Dados foi proposto no âmbito do projeto *Horizon 2020* (H2020) *Production harmonizEd Reconfiguration of Flexible Robots and Machinery* (PERFoRM), a sua definição foi baseada nos requisitos do mesmo. O Modelo de Dados é constituído por 20 classes diferentes, e estas são capazes de abstrair elementos de diferentes camadas arquitetónicas, ou seja, é possível no mesmo Modelo de Dados abstrair elementos tanto do baixo nível (*Shoop-floor*) como do alto nível (*Backbone*).

Aliado ao Modelo de Dados, para expor as várias funcionalidades dos elementos do sistema e ao mesmo tempo garantir uma maior interoperabilidade e plugabilidade, foram implementadas Interfaces Standard. Estas Interfaces vão permitir uma troca de dados de forma transparente e standard.

De forma a validar o trabalho desenvolvido, foi ainda proposta uma implementação preliminar da camada Middleware com o intuito de se ter uma abordagem geral. Este Middleware seguiu as escolhas presentes no H2020 PERFoRM, ou seja, combinou-se o software *WinCC Open Architecture* (WINCC OA) com APACHE CXF.

As Interfaces Standard foram implementadas usando a linguagem de programação Java. Já o Modelo de Dados foi implementado em AutomationML.

Para leitura e escrita de ficheiros aml (que têm por base *Computer Aided Engineering Exchange* (CAEX)), foi necessário desenvolver software adicional, isto é, foi implementado um *Parser* de forma a possibilitar a leitura e escrita dos dados. Além do *Parser* foi definida uma Interface Standard para as interações com os ficheiros aml. Para a implementação do *Parser* e da Interface foi utilizada também Java.

1.4 Principais Contribuições

As princípais contribuições do trabalho apresentado neste documento consistem na implementação de um Modelo de Dados Comum genérico e na definição das Interfaces Standard.

O Modelo de Dados desenvolvido é capaz de abstrair diferentes elementos heterogéneos presentes nas várias camadas de um Sistema de Produção (baixo nível e alto nível).

Já as Interfaces Standard vão permitir expor as funcionalidades dos vários elementos e contribuem para uma interligação dos mesmos de uma forma transparente e standard e ainda habilitam a plugabilidade do próprio sistema.

O trabalho apresentado está inserido no âmbito do projeto H2020 PERFoRM. Este trabalho incluiu a organização e participação de um workshop internacional em Bragança (no Instituto Politécnico de Bragança (IPB)). Relativamente a este trabalho este workshop teve como objetivo a apresentação das Interfaces Standard, do AutomationML, do Modelo de Dados desenvolvido e como instanciar o trabalho num sistema real.

Este workshop, em colaboração com o IPB, o grupo Loccioni (LOC) e a *University of Applied Sciences Hochschule Emden/Leer* (HSEL), levou à submissão de uma publicação com o nome *Integration and Deployment of a Distributed and Pluggable Industrial Architecture for the PERFoRM Project* (Angione et al., 2017) para a *27th International Conference on Flexible Automation and Intelligent Manufacturing* (FAIM) (FAIM, 2016). Este documento foi aceite e apresentado em Junho de 2017.

Capítulo 2

Estado de Arte

Nas últimas décadas o mercado global sofreu inúmeras mudanças, avançando da produção em massa, definida por Henry Ford como produção de produtos diversificados em larga escala e a preços relativamente baixos em linhas de montagem, para uma produção de alta qualidade, de grande customização e com ciclos de vida mais curtos. Estas novas exigências por parte dos consumidores levaram as indústrias a mudar, principalmente em termos de qualidade, agilidade e flexibilidade para se conseguirem manter no mercado (Leitão, 2009; Ribeiro & Barata, 2011).

Para resolver os problemas associados a estas mudanças surgiram novos paradigmas na literatura que abrangem diferentes áreas. Se o foco for a produção temos paradigmas como Sistemas Flexíveis da Manufatura (FMS) (Jahromi & Tavakkoli-Moghaddam, 2012; Jovanovi, 2015) e Sistemas Reconfiguráveis da Manufatura (RMS) (ElMaraghy, 2005; Ribeiro & Barata, 2011). Outra área em que apareceram bastantes paradigmas emergentes foi em inteligência artificial e computacional onde temos os Sistemas Holónicos da Manufatura (HMS) (Babiceanu & Chen, 2006; Barbosa, 2015; Koestler & Arthur, 1968), Sistemas Biónicos da Manufatura (BMS) (Ueda, 1992; Ueda, Hatono, Fujii & Vaario, 2000) e Sistemas Evolutivos de Produção (EPS) (Barata, Santana & Onori, 2006; Onori & Barata, 2009; Ribeiro, Barata & Pimentão, 2011).

Cada um destes paradigmas tem um foco diferente, por exemplo os Sistemas Flexíveis de Manufatura focam-se, como o nome indica na flexibilidade do sistema, isto é, permite uma produção dentro do mesmo sistema diversificada e aceita mudanças na produção (Ribeiro & Barata, 2011). Já os Sistemas Reconfiguráveis de Manufatura, focam-se na habilidade de reconfiguração do sistema, ou seja, aceitam alterações tecnológicas ou adições/remoções na produção (ElMaraghy, 2005).

Já os Sistemas Evolutivos de Produção apostam num sistema que evolua com o tempo, e que vá aprendendo. Pretende-se com este paradigma que o sistema se consiga ajustar

de forma autónoma às exigências impostas (Ribeiro et al., 2011).

Associado a estes paradigmas surgiram também bastantes tecnologias emergentes, como por exemplo a Internet das Coisas (IoT) (*Internet of Things*) (F. Chen et al., 2015; Fei Tao, Ying Cheng, Li Da Xu, Lin Zhang & Bo Hu Li, 2014), *Big Data* (M. Chen, Mao & Liu, 2014), Redes de Sensores Wireless (*Wireless Sensor Networks*) (Qiu et al., 2006), Computação em Nuvem (*Cloud Computing*) (Marston, Li, Bandyopadhyay, Zhang & Ghalsasi, 2011; X. V. Wang & Xu, 2013; Xu, 2012). Estas tecnologias começaram a ser introduzidas na manufatura, e assim estamos cada vez mais a caminhar para uma quarta revolução industrial (S. Wang, Wan, Li & Zhang, 2016).

2.1 Indústria 4.0

De forma a combater as exigências dos consumidores, um conjunto de grupos de várias áreas propôs um novo conceito e com o apoio do governo alemão (Wan, Cai & Zhou, 2015) em 2011 foi proposta uma estratégia chamada Indústria 4.0 como parte do *High-Tech Strategy 2020 Action Plan*, o que levou a inúmeras publicações académicas, artigos e conferências (Hermann, Pentek & Otto, 2016). Outros países também propuseram estratégias similares como os Estados Unidos da América com *Industrial Internet* ou a China com *Internet* + (S. Wang et al., 2016). A Indústria 4.0 é vista como a quarta revolução industrial, referida anteriormente.

Como afirmado em (Schuh, Potente, Wesch-Potente, Weber & Prote, 2014) a necessidade do aumento de produtividade é a maior motivação para o acontecimento de uma revolução industrial. Na figura 2.1 são apresentadas sucintamente as várias Revoluções Industriais.

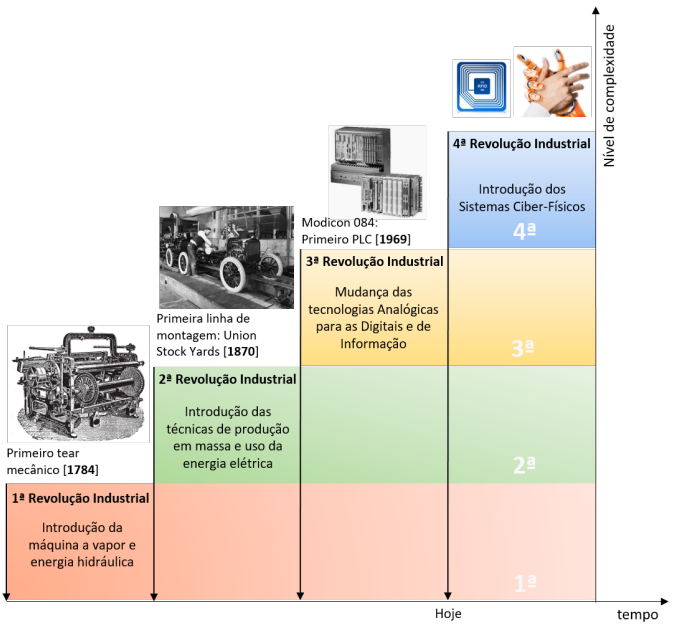

Figura 2.1: História das Revoluções Industriais (adaptado de (Ing Reiner Anderl, 2014))

Nas primeiras três o maior foco foi ao nível do *shop-floor* e nos processos de produção. Concretamente, na primeira foi introduzida a máquina a vapor e a energia hidráulica, já na segunda a maior contribuição foram as técnicas de produção em massa e o uso de energia elétrica. Na terceira Revolução Industrial destaca-se a mudança das tecnologias analógicas para as digitais e as tecnologias de informação (Ing Reiner Anderl, 2014).

Ao contrário das anteriores espera-se uma quarta revolução mais abrangente, pois além da produção em si, foca-se também em outras áreas como nos processos de engenharia. Assim o maior potencial para o aumento da produtividade reside no melhoramento dos processos de decisão (Schuh et al., 2014). É nesta fase que são introduzidos os Sistemas Ciber-Físicos (CPS) (*Cyber-Physical Systems*).

2.1.1 Componentes da Indústria 4.0

Em (Hermann et al., 2016) são evidenciados os quarto maiores componentes na Indústria 4.0, sendo eles: Sistemas Ciber-físicos (CPS), Internet das coisas (IoT), Internet dos Serviços (IoS) (*Internet of Services*) e Fábricas Inteligentes (*Smart Factories*).

Nas próximas subsecções estes componentes vão ser descritos em pormenor.

Sistemas Ciber-físicos (CPS)

Um dos pontos fundamentais na Indústria 4.0 é a fusão entre o mundo digital e físico (Hermann et al., 2016). Esta fusão é conseguida através dos Sistemas ciber-físicos que são descritos por Lee como a integração dos processos físicos e computacionais, onde as redes e computadores integrados controlam e monitorizam os processos físicos, onde estes processos através de um loop de feedback afetam a computação e vice-versa (Lee, 2008).

Olhando para a evolução dos sistemas ciber-físicos é possível distinguir três gerações. Na primeira geração são introduzidas as tecnologias de informação como tags RFID que permitem identificações únicas, na segunda geração os CPS são equipados com sensores e atuadores, já na terceira os CPS tornam-se compatíveis com a rede e são capazes de armazenar e analisar dados (Hermann et al., 2016).

Um exemplo de um sistema ciber-físico é a caixa inteligente *iBin* por Würth. Esta contém um módulo com uma câmara de infravermelhos para controlo de partes C (definidas como partes com importância secundária para o produto final, como parafusos por exemplo (Wurth Group, s.d.)), que vai determinar a quantidade de partes C existentes no *iBin*. Se o stock destas partes for abaixo de um nível mínimo (stock mínimo) o *iBin* tem a capacidade de encomendar mais partes C através de RFID. Assim temos um controlo de consumo de partes C em tempo real (Hermann et al., 2016).

Internet das Coisas (IoT)

A Internet das Coisas é semanticamente composta por dois termos, o que desde o inicio levou a duas abordagens distintas. Por um lado, uma visão orientada à Internet/

rede, enquanto por outro uma abordagem focada em objetos genéricos integrados numa plataforma comum. O resultado de juntar estes dois termos, "Internet" e "Coisas" é um novo nível de inovação (Atzori, Iera & Morabito, 2010).

A Internet das Coisas pode ser definida como uma rede mundial de objetos interligados que são exclusivamente endereçados através de protocolos standard de comunicação (EPoSS, 2008).

Através da IoT, "coisas" e "objetos", como sensores, atuadores, dispositivos móveis, entre outros, interagem e cooperam entre si, para atingirem um objetivo comum. Considerando a definição anteriormente dada para Sistemas Ciber-físicos é possível assumir que "coisas" e "objetos" são na realidade CPS. Assim a IoT pode ser definida como uma rede de CPS, que interagem e cooperam entre si através de esquemas de endereço único (Hermann et al., 2016).

A IoT pode ser aplicada em diversas áreas, como Manufatura (Fei Tao et al., 2014), Agricultura (TongKe, 2013), Saúde (Yang et al., 2014), entre outras. A partir deste conceito, surgiram muitos mais como por exemplo Fábricas Inteligentes, Cidades Inteligentes, Casas Inteligentes, entre outras. Segue-se alguns exemplos existentes onde foi aplicado o conceito da Internet das Coisas:

BigBelly Smart Waste and Recycling System – O *Bigbelly* é um Sistema de gestão de resíduos, é modular e fornece quer informações históricas quer informações em tempo real através de um sistema na nuvem (*cloud system*). Permite assim uma recolha de lixo inteligente, evitando transbordos e acumulação, e tem a habilidade de gerar outras notificações para ser possível manter cidades mais limpas (Rahul, 2017).

CitySense – O *CitySense* é um sistema wireless inteligente de controlo de luzes exteriores. Tem diversas funcionalidades como a adaptação da luz, desta forma reduz a eletricidade consumida e ajusta o brilho das luzes de rua baseando-se na presença de automóveis ou peões e é inteligente ao ponto de filtrar interferências como animais ou árvores (Rahul, 2017).

Internet dos Serviços (IoS)

Buxmann (Buxmann, Hess & Ruggaber, 2009) afirma que a Internet dos serviços vai permitir basicamente que os fornecedores de serviços os forneçam através da internet. Diz ainda que a IoS não é apenas constituída por serviços, mas inclui também as infraestruturas dos serviços, todos os fornecedores/participantes e ainda os modelos de negócio. A IoS permite adicionar mais valor aos serviços, pois vai habilitar a combinação dos serviços que são disponibilizados por diferentes fornecedores e são oferecidos por diferentes vias aos utilizadores e consumidores.

Uma possibilidade de implementação da IoS é a abordagem de Arquitetura Orientada a Serviços (SOA) (*Service Oriented Architecture*) (Picard, Anderl & Schützer, 2013). Este tipo de arquiteturas permite abstrair completamente dos detalhes de implementação sendo que as funcionalidades de um serviço são definidas a partir de uma interface (Bohn, Bobek & Golatowski, 2006). Vários conceitos e tecnologias associados ao SOA

foram surgindo para tentar resolver os problemas de interoperabilidade, como *Java Intelligent Network Infrastructure* (JINI) (Arnold, 1999), *Universal Plug and Play* (UPnP) (Miller, Nixon, Tai & Wood, 2001), *Web Services* (WS) (Sheng et al., 2014), *Device Profile for Web Services* (DPWS) (Bobek, Zeeb, Bohn, Golatowski & Timmermann, 2008; Zeeb, Bobek, Bohn & Golatowski, 2007) e *Open Platform Communications Unified Architeture* (OPC UA) (Hannelius, Salmenperä & Kuikka, 2008; Leitner & Mahnke, 2006), entre outros.

Os dois tipos de serviços web mais populares atualmente são os serviços web baseados em *Simple Object Access Protocol* (SOAP) ou serviços web RESTful. Os serviços web baseados em SOAP assentam basicamente em *Web Services Desciption Language* (WSDL) (Christensen, Curbera, Meredith & Weerawarana, 2001) e em SOAP (Gudgin et al., 2007) enquanto os serviços web RESTful baseiam-se em *Representational State Transfer* (REST) (Fielding, 2000).

A ideia da Internet dos Serviços foi primeiramente implementada num projeto com o nome de *SMART FACE* no programa denominado por *Autonomics for Industrie 4.0*. Com o objetivo de mudar a indústria automóvel, este projeto desenvolveu um sistema de controlo distribuído com base numa arquitetura orientada a serviços, permitindo assim o uso de estações de montagem modulares que vai garantir uma flexibilidade enorme quer na modificação quer na expansão das mesmas. Os veículos automatizados e as estações vão disponibilizar os seus serviços através da IoS (Hermann et al., 2016).

Fábrica Inteligente (*Smart Factory*)

Fábricas Inteligentes ou *Smart Factories* são um fator chave para se avançar para a Indústria 4.0 (Kagermann, Wahlster & Helbig, 2013). A definição de Fábrica Inteligente é uma estrutura "consciente" do seu espaço e que auxilia humanos e máquinas ao longo das tarefas que vão sendo desempenhadas. Esta capacidade é conseguida através de mecanismos que funcionam em background denominados de *Calm-systems*. Estes sistemas em segundo plano, possuem informação contextual e é através de todas as informações que recebem quer do mundo físico (informações como posições ou condições dos produtos e/ou componentes) quer do mundo digital (dados como documentos eletrónicos ou modelos de simulação) que conseguem atingir os seus objetivos (Hermann et al., 2016). No contexto de Fábrica Inteligente, os *calm-systems* apenas se referem ao hardware existente na fábrica e este tipo de sistemas diferenciam-se dos restantes pelo fato de terem a habilidade de comunicar e interagir com o seu meio ambiente (Lucke, Constantinescu & Westkämper, 2008).

Tendo em consideração todas as definições dadas anteriormente, uma Fábrica Inteligente segundo Herman é definida como, uma fábrica constituída por CPS que auxiliam os humanos e as máquinas na execução das suas tarefas para atingir o objetivo pretendido. A comunicação que ocorre entre os vários CPS é através da Internet das Coisas (Hermann et al., 2016).

WITTENSTEIN é um exemplo de um Fábrica Inteligente. Esta está situada na Alemanha e organiza-se segundo os princípios de produção lean. Esta fábrica usa transportadores de peças inteligentes que aquando da chegada de uma peça pronta a ser retirada estes transportadores avisam para tal. Se houver necessidade estes transportadores permitem o inicio do processo novamente. Assim é possível reduzir o número de processos e a carga sobre os operadores (Hermann et al., 2016).

2.1.2 Princípios base da Indústria 4.0

Apesar de existirem diferentes definições de Indústria 4.0 na literatura, em (Hermann et al., 2016) é feita uma revisão da literatura existente e apresentam a sua definição de Indústria 4.0. Hermann ainda identifica os princípios base que são constituintes de qualquer abordagem para a Indústria 4.0. Estes princípios provêm da união dos princípios de cada um dos componentes da Indústria 4.0, anteriormente descritos. Como referido na definição de cada componente temos como princípios base dos *Cyber-Physical Systems* a interoperabilidade, a virtualização e a descentralização. A *Internet of Things* apenas tem como princípio a interoperabilidade, já a *Internet of Services* além da interoperabilidade possui modularidade e é orientada a serviços. Por fim as *Smart Factories* são caracterizadas pelos seguintes princípios: interoperabilidade, virtualização, descentralização e capacidade de tempo real. A tabela 2.1 obtêm-se através da compilação de todos estes princípios.

Tabela 2.1: Princípios chave presentes numa abordagem para a Indústria 4.0 (adaptado de (Hermann, Pentek & Otto, 2016))

	Sistemas Ciber-Físicos	Internet das Coisas	Internet dos Serviços	Fábrica Inteligente
Virtualização	✓	-	-	✓
Descentralização	✓	-	-	✓
Capacidade de Tempo Real	-	-	-	✓
Orientação a Serviços	-	-	✓	-
Modularidade	-	-	✓	-
Interoperabilidade	✓	✓	✓	✓

Todos estes princípios vão ser explicados nas próximas subsecções.

Virtualização

Este princípio diz que os Sistemas Ciber-físicos têm a capacidade de monitorizar o ambiente que os rodeia, ou seja, conseguem monitorizar todos os objetos e processos físicos. A complementar esta capacidade os CPS possuem ainda a habilidade de simular e de criar virtualmente uma cópia do mundo físico (Hermann et al., 2016).

Em casos em que ocorram falhas ou imprevistos o humano pode ser notificado e poderão ser apresentadas várias informações, como por exemplo todos os próximos passos ou medidas de segurança necessárias (Hermann et al., 2016).

Descentralização

A descentralização como um dos princípios da Indústria 4.0 vai garantir aos CPS que consigam trabalhar de forma independente e autónoma, e apenas em caso de falhas nas tarefas ou conflitos nos objetivos os CPS são levados a comunicar e a delegar o problema a níveis superiores de hierarquia no sistema (Hermann et al., 2016).

A descentralização vai criar um ambiente de produção muito mais flexível do que sistemas centralizados, mas considerando assuntos como rastreabilidade e controlo de qualidade é necessário continuar a ter uma visão sobre todo o processo em qualquer momento (Martin, 2017).

Capacidade de Tempo Real

A capacidade de aquisição e análise de dados em tempo real é uma habilidade obrigatória para criar uma fábrica inteligente, pois esta tem de ser capaz de se adaptar e tomar decisões em tempo real aquando a obtenção de *feedback*. Por exemplo, ao surgir uma falha na linha de produção, a fábrica sendo inteligente tem de ser capaz de a detetar e delegar as tarefas para outras máquinas que estejam aptas para tal, ou em caso de não haver solução avisar o humano para tal situação (Hermann et al., 2016; Martin, 2017).

Orientação a Serviços

Os serviços disponibilizados pelas empresas, CPS e humanos, são oferecidos através da Internet dos Serviços, prontos a ser usados por outros utilizadores, sejam eles máquinas ou humanos. Estes serviços podem ser disponibilizados ao mundo ou apenas dentro de uma empresa/rede (Hermann et al., 2016).

Devido à produção ter de ser orientada ao cliente de maneira a criar produtos com as especificações do mesmo, a Internet dos Serviços torna-se essencial pois só assim vai ser possível a existência de uma ligação eficiente entre todas as partes participantes no processo (Hermann et al., 2016; Martin, 2017).

Modularidade

Devido à enorme velocidade de mudanças nos produtos e nos requisitos impostos pelos consumidores, é cada vez mais difícil haver tempo útil suficiente para a adaptação das linhas de produção, e é neste ponto que este princípio se torna essencial. Um sistema modular é um sistema aglomerado por vários módulos individuais. Devido a esta característica este tipo de sistemas torna-se bastante flexível e permite rápidas mudanças e adaptações a alterações nas características dos produtos, mudanças nos requisitos dos mesmos, e até às flutuações sazonais existentes. Esta adaptação é possível através de adição, expansão e/ou remoção de módulos (Hermann et al., 2016).

Interoperabilidade

A interoperabilidade é a capacidade de comunicação entre dois sistemas. Na Indústria 4.0 este é um dos princípais fatores, pois a comunicação entre sistemas é uma das suas bases teóricas, sendo este o princípio que torna uma Fábrica Inteligente uma realidade. Na Indústria 4.0 esta comunicação entre empresas, humanos e máquinas é realizada através da Internet das coisas e da Internet dos serviços (Hermann et al., 2016; Martin, 2017).

A importância deste princípio pode ser confirmada na tabela 2.1, pois este é o único que está presente em todos os componentes. A interoperabilidade tornou-se num dos maiores desafios presentes na Indústria 4.0, pois não é fácil a comunicação entre sistemas que são completamente distintos e trabalham com as suas próprias tecnologias. Então como é possível a troca de dados transparente e consistente entre sistemas heterogéneos presentes em camadas de ação diferentes? A resposta é a definição de standards aceites pela indústria (Peres et al., 2016).

Existem projetos europeus como *Batch Process Automation with an Ontology-driven Multi-Agent System* (BatMAS) (Lepuschitz, Lobato-Jimenez, Axinia & Merdan, 2015) e *7th Framework Programme* (FP7) *Plug and produce intelligent multi-agent environment based on standard technology* (PRIME) (Rocha et al., 2014) que já fizeram progressos no caminho da standarderização, sendo que o FP7 PRIME aborda a importância de uma linguagem semântica e uma representação de dados comum para existir interoperabilidade e plugabilidade, devido a quantidade de sistemas heterogéneos e das complexidades desses sistemas de manufatura inteligentes (Orio, Rocha, Ribeiro & Barata, 2015; PERFoRM Project, 2017).

2.2 Modelação e Representação de dados

Como anteriormente referido, um dos pontos fulcrais para a Indústria 4.0 é a interoperabilidade e a troca de dados de forma segura e transparente. Este ponto é bastante importante devido à necessidade de interligar vários elementos do sistema já existentes, como Controladores Lógicos Programáveis (PLC), robots, softwares de bases de dados, entre outros. Para esta interoperabilidade ser assegurada é necessário a utilização de interfaces standard e estas devem adotar um modelo de representação de dados comum (Peres et al., 2016).

Como referido por Spyns (Spyns, Meersman & Jarrar, 2002), apenas a estrutura e a integridade dos conjuntos de dados são especificados nos modelos de dados, e a construção destes modelos vai depender das necessidades especificas das tarefas e dos objetivos do projeto/empresa. A semântica dos modelos de dados funciona como um acordo entre as duas partes participantes, sendo elas, os programadores e os utilizadores do modelo de dados, logo estes modelos precisam ser muitas vezes atualizados *on the fly*, pois sempre que aparece um requisito novo ou existem alterações aos já existentes, é necessário adaptar o modelo de dados para tal. Dois exemplos de tecnologias para construção de modelos

de dados referidas em (Spyns et al., 2002) são as bases de dados ou esquemas *eXtensible Markup Language* (XML).

Ao longo dos últimos anos surgiram vários standards industriais, onde são providenciadas várias definições semânticas para modelação e troca de dados, focando várias áreas na indústria da manufatura. Como exemplo temos o IEC 62264 que se baseia no ANSI/ISA-95 em que há uma plataforma que pretende facilitar a interoperabilidade e integração dos sistemas empresariais e dos sistemas de controlo. Como outros exemplos temos IEC 62714, ISO 15926, entre outros (PERFoRM Project, 2017).

Devido ao aparecimento de tantos standards novos, surgiram consequentemente implementações com as especificações definidas nesses mesmos standards. Estas implementações são baseadas em XML e existe uma grande variedade. Alguns exemplos destas implementações (Peres et al., 2016; PERFoRM Project, 2017) são:

- IEC 61512 BatchML – Implementação em XML do ANSI/ISA-88 que pertence à família de standards para controlo de lotes. Esta ferramenta oferece uma variedade de esquemas XML escritos em *XML Schema Language* (XSD) que vai implementar as especificações do ISA-88;
- IEC 62264 B2MML (ISA-95 2015) – Também escrito em XSD esta ferramenta implementa os standards ANSI/ISA-95 para integração de sistemas de *Enterprise-control* via XML;
- ISO 15926 XMplant – Esta ferramenta oferece acesso à informação dos processos de forma neutra, cumprindo as especificações ISO 15926. Suporta estruturas, atríbutos, geometria dos esquemáticos e modelos 3D;
- IEC 62424 CAEX – CAEX (Schleipen, Drath & Sauer, 2008) é um formato de dados neutro baseado em XML e orientado a objetos que permite a descrição de informação de objetos como por exemplo estruturas hierárquicas;
- IEC 62714 AutomationML – Formato baseado em XML que foi construído em cima de outros standards já bem definidos de várias áreas da engenharia e que tem como objetivo interligar estes formatos. Usa CAEX como base para conseguir estruturação hierárquica, usa *COLLAborative Design Activity* (COLLADA) para a geometria e cinemática, e por fim PLCopen XML para aplicações de controlo (Faltinski, Niggemann, Moriz & Mankowski, 2012).
- *OPC UA Data Model* – *OPC UA Data Model* define um modelo de dados bastante abstrato que suporta relações/referências entre objetos e múltiplas "heranças". É usado pelo OPC UA para representar diferentes tipos de dados de dispositivos incluído semântica e metadata;
- MTConnect (MTConnect Institure, s.d.) – MTConnect é um standard da manufatura baseado em XML para troca de dados entre o *shop-floor* e aplicações de software

para monitorização e análise. Nestes dados estão incluídos dados de dispositivos, identificação, topologia, e características de design como tamanhos dos eixos, velocidades, etc. O MTConnect possui ainda um conjunto de especificações que assegura a interoperabilidade com OPC UA.

Normalmente existe uma camada central no sistema que é complementada com os modelos de dados genéricos. Esta camada tem o nome de Middleware.

2.3 Middleware

A definição do conceito de middleware, muitas vezes associado a arquiteturas do tipo *Enterprise Service Bus*, pode variar consoante as áreas de aplicação, mas falando na Ciência dos Computadores, este conceito descreve uma camada de software e/ou um componente que permite que várias aplicações e componentes diferentes interajam entre eles. Um middleware pode ser definido em poucas palavras com uma plataforma comum de integração.

Num sistema sem a camada de middleware, para haver comunicação entre os elementos do mesmo, é necessário definir interfaces individuais. Mas se o middleware for parte integrante do sistema, em vez de diferentes interfaces vamos ter apenas uma que é a interface que o middleware vai tratar. Esta solução torna o sistema muito mais flexível que um sistema sem middleware, pois assim novos componentes ou aplicações podem ser integradas no sistema sem a necessidade de definir diferentes interfaces para cada elemento novo. Na figura 2.2 estão representados os dois tipos de sistemas.

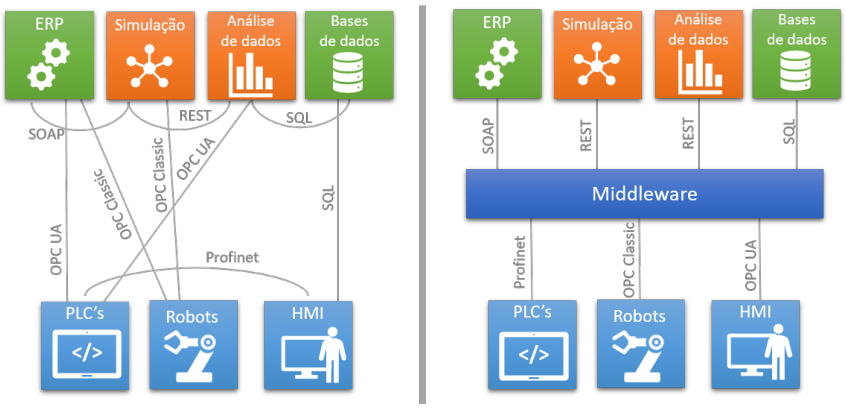

Figura 2.2: Sistema sem Middleware (esquerda) e com Middleware como camada de integração (direita)

Como se pode verificar o sistema da esquerda não possui middleware, o que torna o sistema um tanto caótico, com muitas formas de troca de dados diferentes. Já comparando com o sistema do lado direito, o número de ligações entre elementos diminui consideravelmente, o que torna o sistema muito mais limpo e simples.

Em áreas como a Indústria, onde existem muitas tecnologias diferentes que precisam comunicar e interagir entre elas, as soluções de middleware são uma enorme vantagem. Neste momento com a indústria a atravessar uma grande mudança e os conjuntos de componentes hardware tornarem-se em CPS existe a necessidade destes serem acompanhados por uma camada de software inteligente e interfaces de comunicação. Na indústria ao existirem diferentes elementos desde sensores e máquinas inteligentes até software de planeamento e escalonamento, as soluções middleware são cada vez mais e mais necessárias nos sistemas industriais.

2.3.1 Soluções Existentes

Hoje em dia já existem soluções de middleware com aplicações reais na indústria, alguns exemplos são:

- WINCC OA (ETM, s.d.-a) – Sistema orientado a objetos, criado essencialmente para aquisição e exibição de dados. Contruído com uma abordagem aberta e com bastantes *drivers* definidos, torna a interligação de vários componentes fácil. Assim esta solução é um middleware com potencial.

- *IBM Integration Bus (Manufacturing Pack)* (IBM Corporation, 2016) – *IBM Integration Bus* (IIB) (IBM Corporation, s.d.) é um *Enterprise Service Bus* que vai funcionar como um agente (*Broker*) de integração, que permite a comunicação entre aplicações mesmo através de protocolos de comunicação diferentes. O IIB suporta bastantes protocolos como SOAP, REST e *Hypertext Transfer Protocol* (HTTP) e possui ainda interfaces para diversas bases de dados como Oracle ou *Microsoft SQL Server* e sistemas *Enterprise Resource Planning* (ERP) (SAP, PeopleSoft, etc). É através do modulo *Manufacturing Pack* que é conseguido a integração de sistemas de manufatura, como OPC ou OPC UA.

- *Red Hat JBoss Fuse ESB* (Red Hat, s.d.) – Este *open Enterprise Service Bus* é baseado em Apache ServiceMix (Apache Software Foundation, s.d.), o que permite oferecer todas as funcionalidades de integração presentes no ServiceMix como uma plataforma de integração usando o Apache Camel (Apache Software Foundation, s.d.) e aplicações de integração através de serviços web SOAP ou REST através do Apache CXF (Apache Software Foundation, s.d.). O JBoss Fuse inclui também uma biblioteca com conectores pré-definidos para sistemas já existentes e aplicações standard como bases de dados, web servers, entre outros.

- *SAP MII Plant Connectivity* (SAP, s.d.) – Uma das soluções ERP mais usadas, esta possui uma camada de integração para a manufatura (*Manufacturing Integration and Intelligence* (MII)) que oferece uma plataforma que permite a configuração entre as soluções SAP e componentes externos, como *Manufacturing Execution System* (MES). A *SAP Plant Connectivity* permite que o sistema comunique com componentes no

shop-floor, e as interfaces que o permitem são controladas por agentes. Existem agentes definidos para OPC, OPC UA entre outros.

Devido às exigências dos consumidores, são precisas novas soluções para os sistemas da indústria. Estas soluções passam por modelos de dados genéricos e camadas middleware. Existem algumas opções quer para implementar os modelos de dados quer para criar o middleware, então é preciso comparar o que estes oferecem com as necessidades de cada consumidor.

2.4 Conclusões Gerais

Pelo presente capítulo, podemos verificar que quer a Indústria quer a exigência do consumidor estão a sofrer grandes alterações, como tal as abordagens existentes são cada vez mais inadequadas e obsoletas para todas as necessidades e requisitos atualmente necessários.

Nos últimos anos foram publicados artigos científicos e realizados projetos de investigação, com o objetivo de tentar adaptar os sistemas industriais para o futuro, houve avanços, mas ainda é um tema de difícil aceitação por parte das empresas e existe caminho a percorrer até se conseguir uma solução viável para esta atualização da Indústria atual.

Para tornar os sistemas de manufatura inteligentes e flexíveis como o pretendido na Indústria 4.0, é necessário ter uma linguagem comum de comunicação entre todos os elementos do sistema para ser possível trabalharem em conjunto. Para tal, o desenvolvimento de Modelos de Dados Comuns, soluções Middleware e Interfaces Standard é necessário, mas ainda se estudam soluções para tais abordagens.

Com o estudo realizado, verificou-se que a interoperabilidade dos sistemas/componentes heterogéneos existentes atualmente para formar um sistema inteligente e flexível através de uma linguagem comum ainda está numa fase bastante inicial, existindo assim uma enorme oportunidade de investigação e de trabalho.

Capítulo 3

Arquitetura do PERFoRM

Como referido no capítulo 2, atualmente a indústria está a caminhar para uma quarta revolução, e assim sendo, existe um esforço para a integração das tecnologias emergentes nos sistemas existentes na manufatura e ao mesmo tempo tornar o sistema capaz de suportar as novas exigências dos consumidores. Este suporte é conseguido tornando o sistema inteligente, flexível e dinâmico.

O trabalho desenvolvido enquadra-se no projeto *Horizon 2020 Production harmonizEd Reconfiguration of Flexible Robots and Machinery* (H2020 PERFoRM) (PERFoRM Project, 2016c), e apesar do projeto e da sua arquitetura não serem o foco do documento, para melhor entendimento a próxima secção é uma introdução ao mesmo.

Nas restantes secções o foco vai ser na arquitetura do H2020 PERFoRM onde vão ser introduzidos os requisitos base do projeto e alguns elementos presentes no sistema, importantes no trabalho desenvolvido.

3.1 Visão Geral do PERFoRM

O projeto H2020 PERFoRM tem como objetivo desenvolver e apresentar uma solução tecnologicamente inovadora para lidar com as tendências emergentes na manufatura, como flexibilidade e reconfiguração dos sistemas de produção.

O sistema vai integrar várias ferramentas software de alto nível como Escalonamento, Simulação, Modulação e Suporte de Decisão Inteligente.

A validação deste projeto vai ser realizada em quatro casos de estudo, abrangendo várias áreas da Indústria Europeia como Eletrodomésticos (Whirlpool), Aeroespacial (GKN), Micro veículos elétricos (*Interactive Fully Electrical Vehicles* (I-FEVS)) e produção de compressores de grandes dimensões (Siemens) (PERFoRM Project, 2016a).

O caso de estudo da Siemens vai focar-se maioritariamente em três áreas, sendo elas:

Planeamento e Logística, Produção e Manutenção. O objetivo neste caso de estudo é a identificação de perturbações na linha de produção o mais cedo possível e fornecer todos os dados possíveis aos parceiros responsáveis (PERFoRM Project, 2016a).

Já no caso de estudo da I-FEVS o foco é apenas em duas áreas sendo elas a Logística e Produção. O objetivo é assegurar a alta qualidade nos produtos desenvolvidos. Atualmente os processos existentes são em parte manuais e com o projeto espera-se que a linha de produção se torne mais automatizada e flexível, para fácil produção de veículos com diversas configurações (PERFoRM Project, 2016a).

O objetivo no caso de estudo da Whirlpool é o estabelecimento de um sistema de monitorização em tempo real que deve ser capaz de correlacionar os *Key Perfomance Indicator* (KPI) com os *Key Business Factor* (KBF) (PERFoRM Project, 2016a).

Por último, o caso de estudo da GKN tem como objetivo criar um ambiente de produção baseado em micro células de produção. Com esta abordagem é garantido ao sistema uma enorme flexibilidade e reconfigurabilidade.

3.2 Arquitetura do PERFoRM

Como referido em (PERFoRM Project, 2016b) a arquitetura do PERFoRM cobre duas das camadas definidas pelo ISA-95, nomeadamente a camada L2 e L3, ou seja, a camada de *Supervisory Control* e a *Manufacturing Operations Management* respetivamente. Esta arquitetura vai oferecer uma infraestrutura e a metodologia para implementar a nova geração de sistemas de automação na forma de sistemas distribuídos de automação baseados em *cloud*, seguindo todos os princípios definidos na Indústria 4.0, sendo eles a interoperabilidade, virtualização, descentralização, capacidade de tempo real, orientação a serviços e modularidade, como anteriormente referidos e explicados.

Com base nos vários requisitos de cada caso de estudo e os resultados de projetos anteriores bem sucedidos, descritos em (PERFoRM Project, 2016b), a arquitetura deve ter em consideração alguns pontos fundamentais:

- O sistema tem de ser baseado em componentes de produção inteligentes e heterogéneos;
- O sistema deve ser capaz de suportar uma reconfiguração segura e transparente;
- O sistema deve ser capaz de realizar simulações e melhorar as mesmas. Deve ser capaz de melhorar ainda o próprio planeamento e características operacionais;
- O sistema deve ter a habilidade de fornecer ao operador humano toda a informação relevante e assistência ao mesmo.

A arquitetura é baseada numa rede de dispositivos Hardware distribuídos e de aplicações Software, onde as suas funcionalidades são expostas como serviços seguindo a abordagem de SOA, onde são conectados de maneira transparente usando um Middleware Industrial. A arquitetura está ilustrada na figura 3.1.

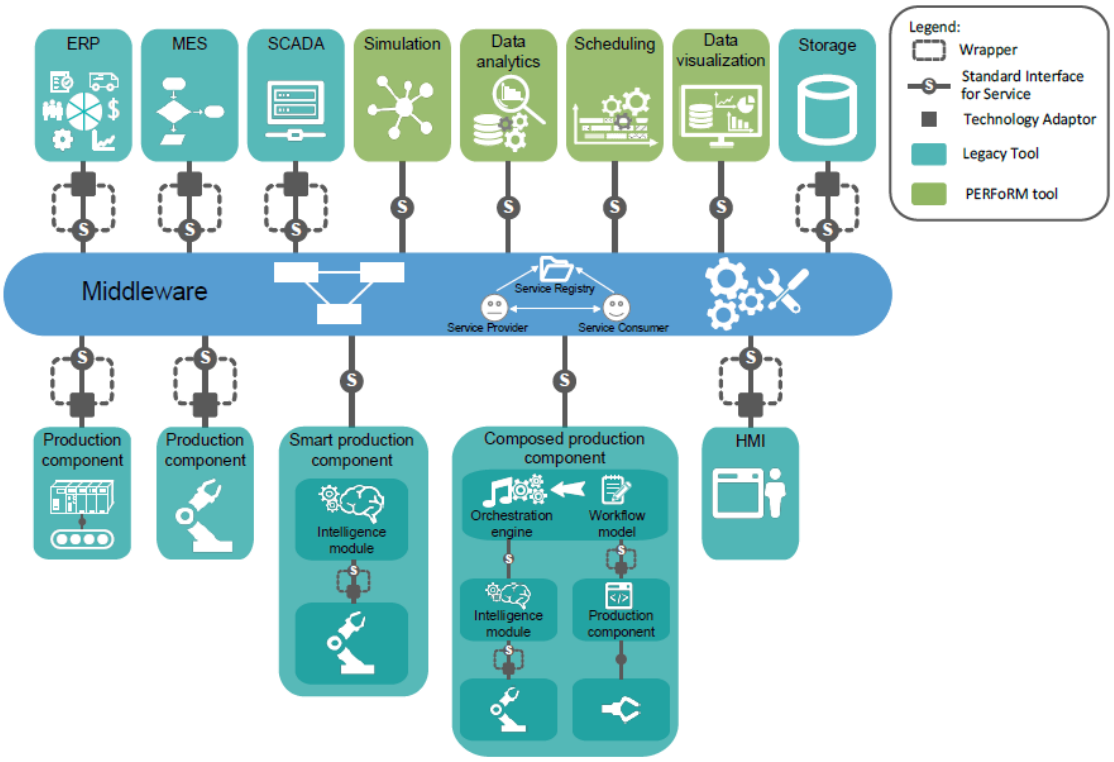

Figura 3.1: Visão geral da arquitetura do PERFoRM (PERFoRM Project, 2016b)

Como se pode verificar na figura 3.1, a arquitetura pode ser dividida, em 3 camadas. No topo existe a camada de alto nível, onde se encontram as aplicações software como a ferramenta de Simulação ou Análise de Dados. No centro tem-se a camada de Middleware que vai ser o elo de ligação entre todos os elementos presentes no sistema. Por fim, tem-se a camada de baixo nível ou o *shop-floor* onde se encontram os elementos hardware, como por exemplo os robots e PLCs.

Apesar de existirem diversos elementos na arquitetura apresentada, neste documento apenas vai ser dada uma descrição dos elementos importantes para o trabalho desenvolvido. Este trabalho foca-se no Modelo de Dados Comum e nas Interfaces Standard e como anteriormente referido estes complementam o Middleware.

Assim sendo, nas próximas subsecções vai ser apresentada uma descrição do Middleware e dos seus constituintes como os Adaptadores Tecnológicos e as próprias Interfaces Standard.

3.2.1 Middleware Industrial

No contexto da Indústria 4.0 onde o projeto se encontra inserido o Middleware é um componente da arquitetura bastante importante pois tem como objetivo assegurar uma interconexão transparente, segura e confiável dos diversos dispositivos heterogéneos, como células robóticas e Controladores Lógicos Programáveis (PLCs) e aplicações software como *Manufacturing Execution System* (MES) e *Supervisory Control And Data Acquisition* (SCADA).

As comunicações e interações através do middleware são uma mais valia, pois os vários elementos presentes num sistema provavelmente não possuem todas as interfaces necessárias para interagir entre si. Com a camada de Middleware presente na arquitetura existe apenas a necessidade de conhecer uma interface, isto é, os vários elementos do sistema apenas necessitam de saber como interagir com o Middleware, pois todas as comunicações entre elementos passam por este.

A existência de apenas uma interface garante uma grande vantagem nestes tipos de sistemas, pois vai permitir que novos componentes e/ou aplicações possam ser facilmente integrados nos sistemas, conferindo assim um grande nível de escabilidade ao sistema.

Outra vantagem do middleware proposto é ser uma abordagem distribuída em *cloud*, em vez de uma abordagem centralizada, pois este tipo de abordagem reduz as limitações de escabilidade que os sistemas centralizados possuem. A não existência de um elemento central protege o sistema contra falhas que poderiam existir neste elemento que incapacitariam todo o sistema (PERFoRM Project, 2016b).

Esta camada de integração lida com a ligação dos componentes heterogéneos de produção através dos princípios de orientação a serviços, isto é, cada componente vai expor as suas funcionalidades como um serviço, que vai ser descoberto e pedido por outros componentes. Esta camada vai permitir uma comunicação quer horizontal quer vertical.

3.2.2 Interfaces Standard

Como referido no capítulo 2, um dos maiores desafios da Indústria 4.0 e consequentemente deste projeto é a interoperabilidade em ambientes industriais reais, devido à imensidade de diferentes componentes existentes nos sistemas de manufatura e muitos deles com representação de dados própria e não compatíveis entre si.

Considerando esse aspeto fundamental, na arquitetura do H2020 PERFoRM foi proposto a utilização de Interfaces Standard para resolver algumas questões como plugabilidade e interoperabilidade, permitindo assim uma comunicação transparente entre dispositivos e aplicações.

Estas Interfaces Standard devem possuir todos os meios para descrever e expor todas as funcionalidades dos dispositivos, aplicações e ferramentas de uma maneira única, standard e transparente de modo a promover assim a interoperabilidade e a plugabilidade, anteriormente referidas.

As Interfaces Standard devem descrever a semântica usada e o fluxo de dados existente, isto é, os *inputs* e *outputs* necessários das várias comunicações existentes.

A abordagem usada no H2020 PERFoRM é uma abordagem orientada a serviços, onde as funcionalidades dos vários elementos do sistema vão ser expostos como serviços. Devido a tal abordagem neste projeto, adota-se um Modelo de Dados Comum que vai servir como formato para troca de dados entre os elementos arquitetónicos do projeto.

Assim como as funcionalidades dos vários elementos do sistema são expostas como serviços, as Interfaces Standard devem oferecer certas funcionalidades relacionadas com a invocação dos mesmos, como a lista de serviços que precisam de ser implementados na interface, o nome, os parâmetros de entrada e de saída dos serviços e ainda a definição do modelo de dados que os serviços lidam.

Um dos focos do trabalho desenvolvido são as interfaces Standard. Como se pode verificar na figura 3.2 existe uma Interface para a ligação do Middleware com o baixo nível e outra para a ligação com o alto nível, assim vão ser definidas duas Interfaces Standard com focos diferentes.

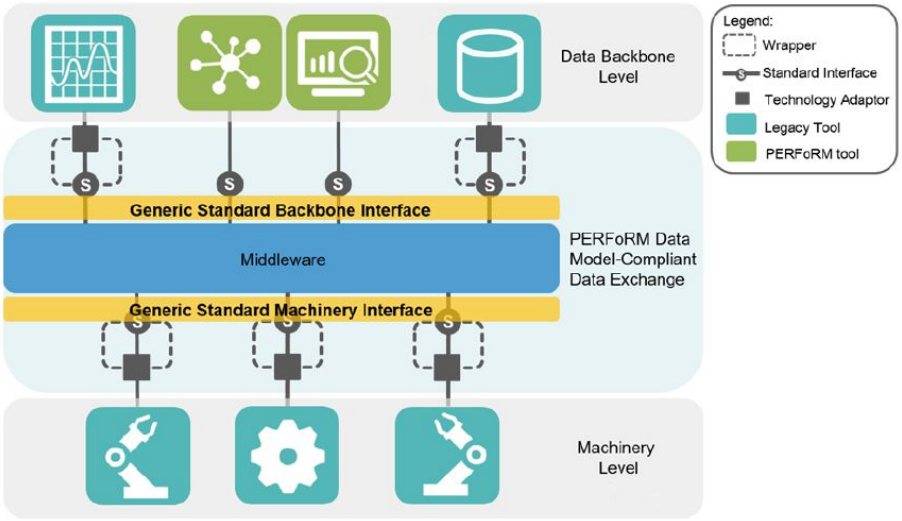

Figura 3.2: Camadas da Arquitetura do PERFoRM (PERFoRM Project, 2016b)

A Interface Standard que existe para a ligação entre o *Shop-floor* e o Middleware vai ter de expor todas as funcionalidades relacionadas com o Hardware, como por exemplo, a ligação e acesso aos vários elementos do *Shop-floor*.

Já a Interface de ligação entre a camada de *Data Backbone* e o Middleware vai ter de expor as funcionalidades de todas as aplicações software presentes na camada de alto nível, como por exemplo, as funcionalidades da ferramenta de Simulação ou de Escalonamento.

Como referido, as Interfaces têm de expor todos as funcionalidades dos diferentes elementos do sistema, e assim foram analisados os requisitos dos vários casos de estudo e das várias aplicações desenvolvidas no projeto, de forma a ser possível a definição das

Interfaces Standard, isto é, saber que parâmetros de entrada são necessários para as várias aplicações ou o resultado das mesmas por exemplo.

A implementação das Interfaces Standard está descrita na secção 5.2.

3.2.3 Adaptadores Tecnológicos

Na Indústria de Manufatura os sistemas existentes (*legacy*) são heterogéneos, onde no nível da maquinaria (*shop-floor*) tem-se robots ou *Human Machine Interfaces* (HMI) e ao nível do *backbone* (alto nível) temos por exemplo bases de dados de produção ou MES. Todos estes elementos têm a sua forma de comunicar e transmitir dados e um modelo de dados próprio, o que dificulta a interligação entre todos eles e a inserção de novos componentes no sistema.

A arquitetura inovadora do PERFoRM só pode ser aceite e usada se houver uma possível integração com estes sistemas. Assim sendo, os adaptadores tecnológicos são uma ferramenta chave para resolver esta integração. Estes adaptadores são o ponto de ligação dos sistemas já existentes ao Middleware do PERFoRM e são os responsáveis por transformar o modelo de dados presente em cada elemento do sistema no modelo de dados definido nas interfaces standard.

Estes adaptadores apenas são necessários quando existe a necessidade de interligação do sistema do H2020 PERFoRM com elementos *legacy*, pois os componentes e aplicações desenvolvidas no âmbito do projeto já têm por base o Modelo de Dados Comum, não sendo necessária assim qualquer tipo de conversão.

Dentro do H2020 PERFoRM vão ser desenvolvidos três tipos de adaptadores tecnológicos, sendo eles: Adaptador Tecnológico Standard, Adaptador Tecnológico de Tempo Real e Adaptador Tecnológico HMI. Cada um destes adaptadores vai ter um foco diferente, definidos no âmbito do projeto, e estes podem ser vistos na figura 3.3.

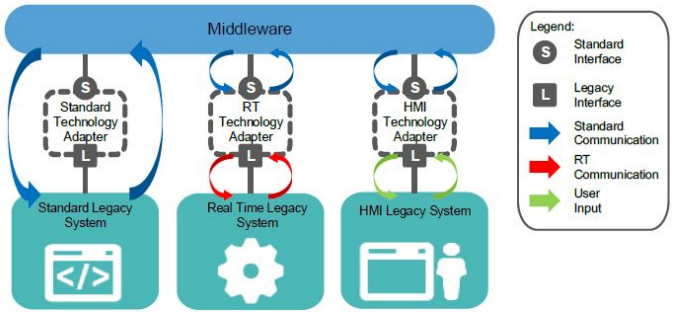

Figura 3.3: Tipos de Adaptadores Tecnológicos (PERFoRM Project, 2016b)

Um dos focos é o tempo real, sendo este bastante importante, por exemplo nas células robóticas, pois estas necessitam de pequenos ajustes e correções dependendo da informação adquirida de sensores existentes. Estes ajustes têm de ser realizados o mais breve possível e assim sendo o adaptador tem de ser extremamente rápido para atender estas necessidades.

Em segundo lugar tem-se os adaptadores HMI, que além de servirem para monitorização e controlo dos recursos da produção, podem ser usados para receber *feedback* dos utilizadores e aprender através das atividades dos mesmos. Estes adaptadores já não precisam de ser tão exigentes a nível temporal como os anteriores.

Por fim tem-se os adaptadores tecnológicos standard que focam os elementos que não se encontram dentro das duas categorias anteriores. Estes elementos são por exemplo os PLCs onde as comunicações e trocas de dados, apesar de normalmente abrangerem diferentes tecnologias e protocolos, são standard.

Como anteriormente referido, além dos elementos descritos neste capítulo, é necessário um Modelo de Dados Comum para existir um formato genérico e único para a troca de dados no projeto H2020 PERFoRM.

O maior foco do trabalho desenvolvido foi o Modelo de Dados Comum e este será apresentado e descrito em pormenor no próximo capítulo.

Capítulo 4

Modelo de Dados Comum

Como referido anteriormente, um dos maiores desafios da Indústria 4.0 é a interoperabilidade em ambientes reais da indústria. A complexidade deste desafio provém da necessidade de lidar com a representação e troca de dados de maneira segura e transparente das várias entidades diferentes presentes no sistema.

Como explicado no capitulo 3, para garantir assim a interoperabilidade dos dispositivos heterogéneos e aplicações software, a arquitetura do H2020 PERFoRM adota as Interfaces Standard e um Modelo de Dados Comum.

O objetivo do Modelo de Dados Comum é a existência de um formato de dados genérico, que deve ser usado na troca de informações entre os elementos arquitetónicos do H2020 PERFoRM. O Modelo de Dados deve cobrir todos os aspetos semânticos necessários para cada elemento presentes no sistema.

O grande foco do documento é o Modelo de Dados que vai ser apresentado e descrito em pormenor ao longo do presente capítulo.

4.1 Modelo de Dados Comum

Baseado nos requisitos do PERFoRM anteriormente referidos, foi desenhado um modelo de dados o mais genérico possível de forma a preencher todos os requisitos do sistema e dos vários casos de estudo. O modelo de dados pode ser visto na sua totalidade na figura 4.1.

De maneira a existir apenas um Modelo de Dados, e que este consiga cobrir todos os aspetos, quer de dispositivos hardware quer aplicações software de uma maneira genérica e abstrata, existem classes que pertencem simultaneamente ao alto nível (*Data Backbone*) e ao nível da maquinaria (*Shop-floor*).

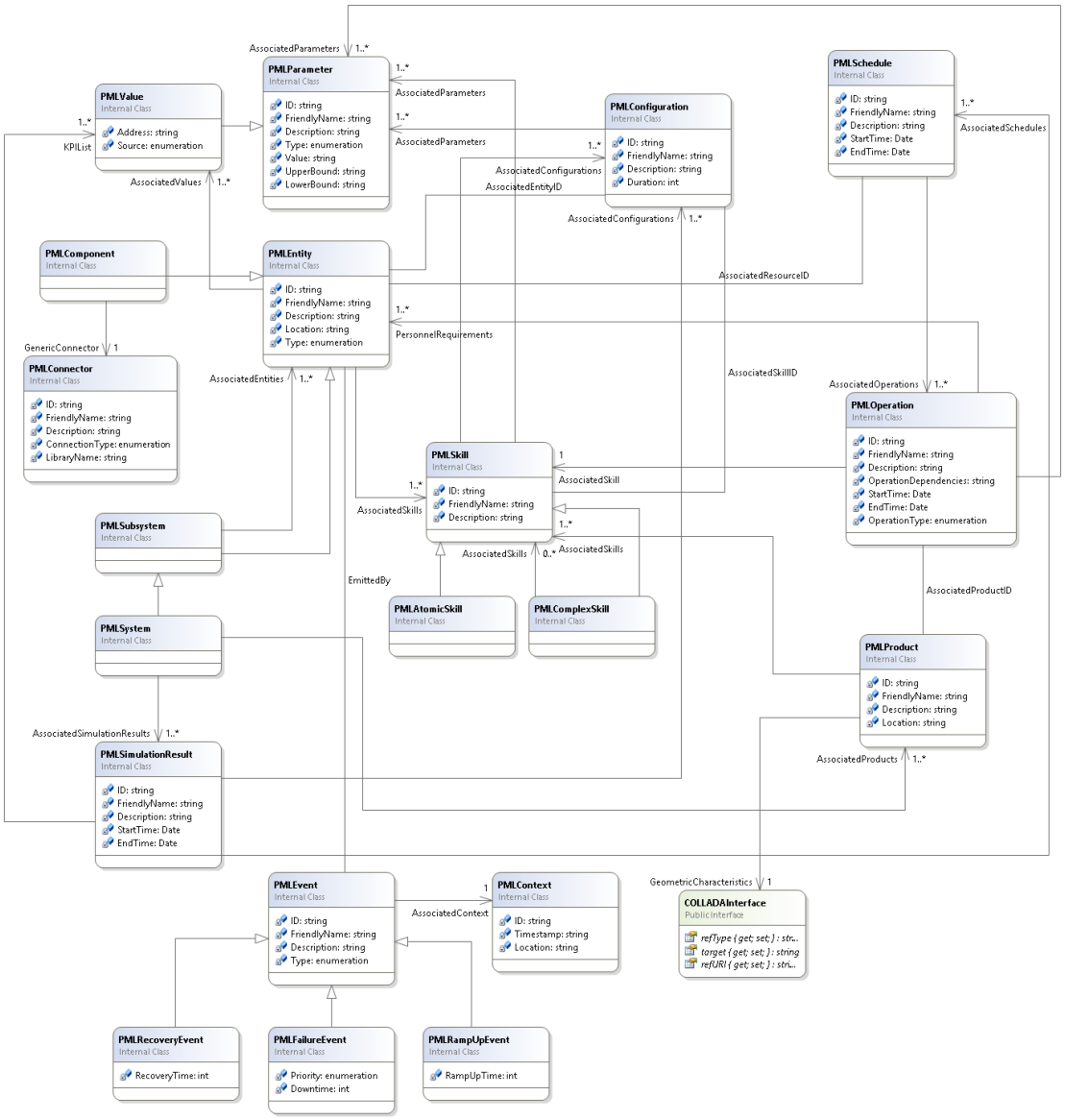

Figura 4.1: Modelo de Dados Comum

Todas as classes presentes no Modelo de Dados vão ser descritos em pormenor nas próximas secções.

4.1.1 PMLParameter e PMLValue

Estes dois elementos vão permitir a representação da informação pertinente referente quer ao nível da maquinaria quer no alto nível. Esta informação pode ser por exemplo: parâmetros usados nas aplicações (como simulação), configurações ou nas operações (funções, habilidades ou tarefas executadas pelos elementos presentes no *shop-floor*), ou ainda dados extraídos do *shop-floor* de diferentes fontes como PLC's ou bases de dados. Estas duas classes podem ser vistas na figura 4.2.

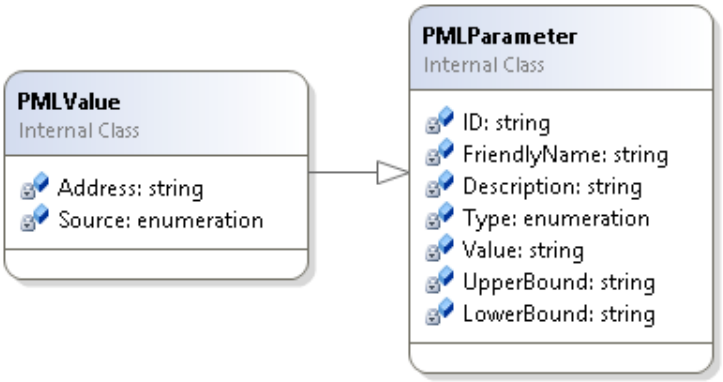

Figura 4.2: PMLParameter e PMLValue

Como se pode verificar existe uma relação de parentesco entre estas duas classes. A classe PLMValue estende a classe PMLParameter.

Os atributos presentes na classe PMLParameter são os seguintes:

- ID – *String* que é a identificação única do elemento;
- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- Type – Enumeração que especifica o tipo de parâmetro ou valor (DOUBLE ou INTENGER por exemplo);
- Value – *String* genérica que contem o valor atual;
- UpperBound – *String* que poderá ser usada para definir o valor mais alto do atributo Value;
- LowerBound – *String* que poderá ser usada para definir o valor mais baixo do atributo Value.

A classe PMLValue vai herdar todos os atríbutos da classe-pai (PMLParameter). Assim além destes o PMLValue possui dois atributos adicionais:

- Address – *String* que identifica o endereço de um valor, por exemplo OPC UA tag ou a key da base de dados, etc;
- Source – Enumeração que descreve de onde o valor poderá ser obtido (PLC ou AGENT por exemplo).

4.1.2 PMLEntity

A classe PMLEntity é uma representação genérica e abstrata de uma entidade. Possui toda a informação comum que existe referente a componentes e subsistemas (conjunto de componentes e possivelmente outros subsistemas que trabalham em conjunto para um objetivo comum).

Esta classe está representada na figura 4.3.

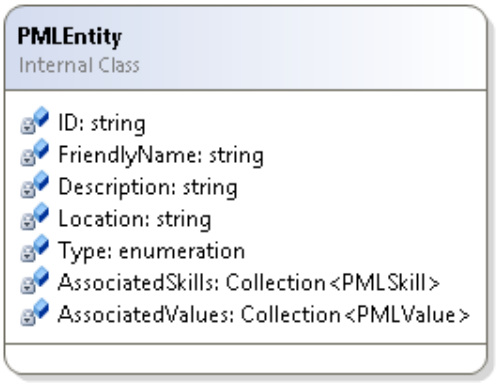

Figura 4.3: PMLEntity

Esta classe não vai ser diretamente usada, mas vai permitir definir coleções genéricas de elementos, podendo ser eles componentes ou subsistemas sem saber à partida qual a composição destas coleções. Na próxima subseção vai ser evidenciada esta situação.

Os atributos presentes neste conceito são:

- ID – *String* que é a identificação única do elemento;
- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- Location – *String* genérica que indica a localização da entidade no *shop-floor*, sendo que esta informação poderá ser um conjunto de coordenadas x/y ou coordenadas GPS, etc.

- Type – Enumeração que especifica o tipo de entidade;
- AssociatedSkills – Coleção de operações que detalham as tarefas que uma certa entidade pode executar, como soldar, apanhar e largar, etc;
- AssociatedValues – Coleção de elementos da classe PMLValue que detalha valores importantes referentes a uma entidade, como *inputs*, *outputs*, etc.

4.1.3 PMLComponent e PMLSubsystem

Como anteriormente referido, de maneira a abstrair de forma genérica os vários elementos do sistema, foi definida a classe PMLEntity, e assim as classes PMLComponent e PMLSubsystem vão estender a classe PMLEntity. Estas classes estão descritas na imagem 4.4.

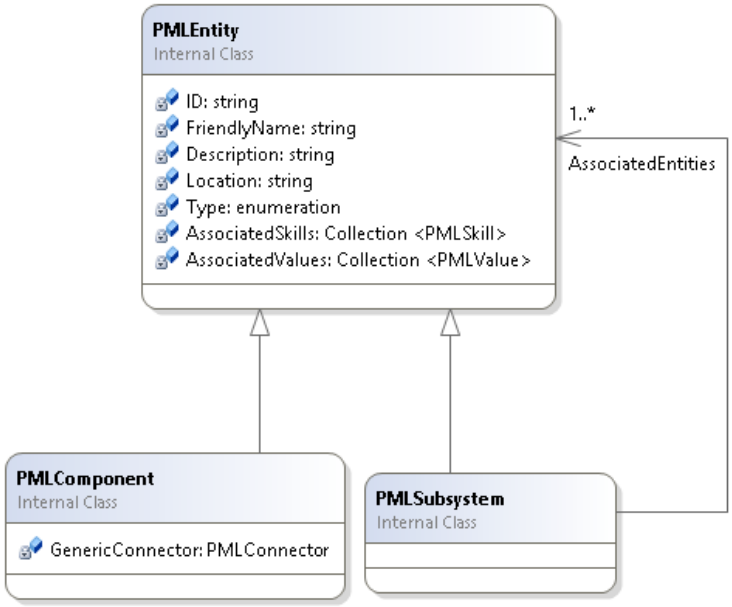

Figura 4.4: PMLEntity, PMLComponent e PMLSubsystem

O PMLComponent representa um elemento único no sistema que pode ter associado determinadas habilidades, como mover ou apanhar, e possui alguns valores que são relevantes para extração, como tempo de ciclo ou consumo energético. A representação é suficientemente genérica para ser possível que um componente não seja apenas um dispositivo físico, mas também recursos virtuais ou até operadores humanos.

Já o PMLSubsystem funciona de forma semelhante, mas pode conter vários componentes e/ou até outros subsistemas. É um elemento recursivo na medida em que estende a class PMLEntity e pode conter PMLEntities na sua composição.

O design genérico permite vários níveis de granularidade usando o mesmo modelo de dados, isto é, podemos ter um sistema onde o robot é o nível mais baixo de abstração

enquanto noutro sistema o mesmo robot pode ser um subsistema dentro do sistema, pois pode encapsular vários elementos como sensores.

Os atributos comuns nestas duas classes, são os atributos herdados da classe-pai e são eles:

- ID – *String* que é a identificação única do elemento;
- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- Location – *String* genérica que indica a localização da entidade no *shop-floor*, sendo que esta informação poderá ser um conjunto de coordenadas x/y ou coordenadas GPS, entre outras;
- Type – Enumeração que especifica o tipo de entidade;
- AssociatedSkills – Coleção de operações que detalham as tarefas que uma certa entidade pode executar, como soldar, apanhar e largar, etc;
- AssociatedValues – Coleção de elementos da classe PMLValue que detalha valores importantes referentes a uma entidade, como *inputs*, *outputs*, etc.

Adicionalmente a class PMLComponent possui:

- GenericConnector – Um PMLConnector que especifica como o componente interage com o *shop-floor*.

E como referido anteriormente, a classe PMLSubsystem tem:

- AssociatedEntities – Coleção de elementos que podem ser das classes PMLComponent ou PMLSubsystem, que fazem parte de um determinado subsistema.

4.1.4 PMLSkill

As tarefas que podem ser executadas por um determinado PMLComponent ou PMLSubsystem são expostas como PMLSkill. Esta classe esta representada na figura 4.5.

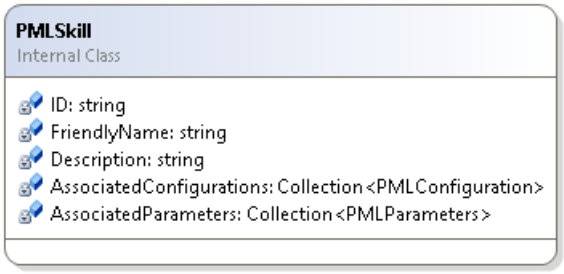

Figura 4.5: PMLSkill

Como os outros elementos presentes neste modelo de dados, esta classe possui alguns atributos que permitem identificar e descrever a mesma. Esta classe funciona como a classe PMLEntity pois é um conceito que não vai ser diretamente usado, mas vai servir para a criação de coleções genéricas de PMLAtomicSkill ou PMLComplexSkill que vão ser descritas a seguir.

Os atributos presentes nesta classe são:

- ID – *String* que é a identificação única do elemento;
- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- AssociatedConfigurations – Coleção de elementos do tipo PMLConfiguration que representam possíveis configurações para a execução de uma determinada habilidade;
- AssociatedParameters – Coleção de parâmetros que representam possíveis parâmetros para uma determinada habilidade.

4.1.5 PMLAtomicSkill e PMLComplexSkill

Conceptualmente as habilidades ou operações podem ser divididas em duas categorias, sendo estas habilidades atómicas ou habilidades complexas, estas são abstraídas no modelo de dados por PMLAtomicSkill e PMLComplexSkill respetivamente. Podemos ver estas classes na figura 4.6.

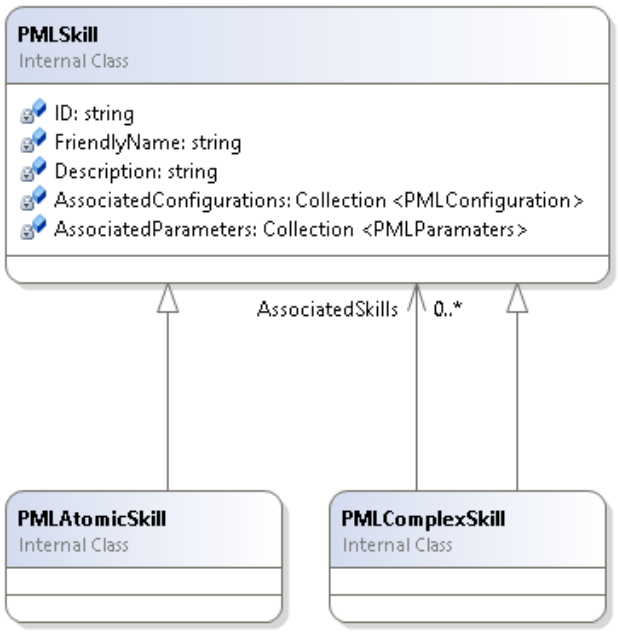

Figura 4.6: PMLSkill, PMLAtomicSkill e PMLComplexSkill

A PMLAtomicSkill é a forma mais simples de uma operação, pois representa apenas uma única ação executada por uma entidade. Não existem atributos adicionais na classe PMLAtomicSkill. Os seus atributos são os herdados da sua classe-pai como se pode verificar na figura 4.6.

Já a PMLComplexSkill consiste numa operação que pode ser definida por uma combinação de várias operações, sendo que estas podem ser atómicas ou complexas, e assim sendo esta classe também é uma classe recursiva. Por esse motivo, existe a adição de um atributo, assim sendo esta é composta pelos seguintes atributos:

- ID – *String* que é a identificação única do elemento;
- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- AssociatedConfigurations – Coleção de elementos do tipo PMLConfiguration que representam possíveis configurações para a execução de uma determinada operação;
- AssociatedParameters – Coleção de parâmetros que representam possíveis parâmetros para uma determinada habilidade;
- AssociatedSkills – Coleção de elementos da classe PMLSkill que podem ser atómicos ou complexos. Esta coleção descreve a composição de habilidades de uma determinada operação complexa.

4.1.6 PMLConfiguration

A PMLConfiguration vai permitir uma definição de possíveis configurações para a execução de uma operação ou de uma determinada simulação, de acordo com um conjunto de parâmetros. Podemos ver esta classe na figura 4.7.

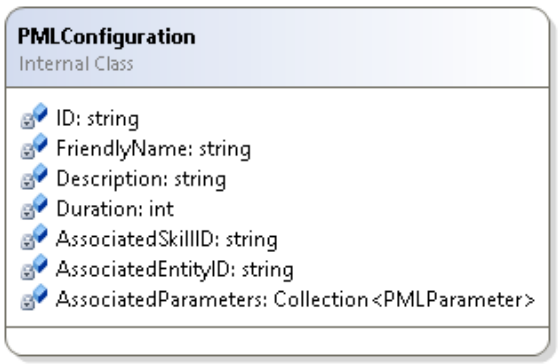

Figura 4.7: PMLConfiguration

A existência desta classe, permite que outras classes (PMLSkill e PMLSimulationResult) tenham bem definidas as várias configurações possíveis para a execução de uma determinada tarefa.

Este conceito é composto pelos seguintes atributos:

- ID – *String* que é a identificação única do elemento;
- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- Duration – Tempo que indica a duração que uma habilidade demora a ser executada numa determinada configuração;
- AssociatedSkillID – *String* que contém o identificador único da habilidade associada;
- AssociatedEntityID – *String* que contém o identificador único da entidade associada;
- AssociatedParameters – Coleção de elementos da classe PMLParameters, que representam os possíveis parâmetros associados a uma certa configuração, que permite parametrizar diferentes configurações.

4.1.7 PMLProduct

Este conceito abstrai o tipo de um determinado produto e as suas caraterísticas base, para permitir um processo orientado à descrição do produto. A classe PMLProduct pode ser vista na figura 4.8.

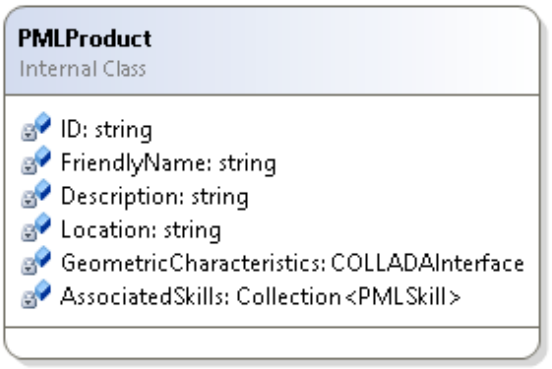

Figura 4.8: PMLProduct

O produto como os outros elementos do Modelo de Dados, possuí um identificador único. Esta classe contém uma descrição de todas as operações necessárias para a produção de um determinado tipo de produto e possui ainda um *link* para uma descrição

externa das suas caraterísticas geométricas e cinemáticas. Para esta descrição é usado o formato *COLLAborative Design Activity* (COLLADA), que é um esquema XML standard para trocas de recursos 3D entre as várias aplicações software.

Este conceito tem os seguintes atributos:

- ID – *String* que é a identificação única do elemento;
- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- Location – *String* que indica a localização do produto para questões de rastreabilidade, pode ser por exemplo um par de cordenadas x/y ou um ID de um recurso ou estação, consoante a semântica definida;
- GeometricCharateristics – Uma COLLADAInterface que é um elemento definido no AutomationML, para conectar o PMLProduct a um ficheiro externo do tipo COLLADA contendo as caraterísticas geométricas e cinemáticas do produto;
- AssociatedSkills – Coleção de elementos do tipo de PMLSkill que vão descrever as várias operações necessárias para a produção de um determinado produto.

4.1.8 PMLConnector

A classe PMLConnector vai ter a descrição de toda a informação necessária para a comunicação entre um componente e o *shop-floor*. Esta classe é uma abstração genérica que permite que o modelo de dados tenha aplicabilidade em diferentes sistemas, apesar da grande variedade de protocolos de comunicação. Como se pode verificar na figura 4.9 o PMLConnector faculta a identificação do tipo de comunicação, como por exemplo OPC UA, e permite a descrição da interface que deve ser usada para a comunicação com o componente.

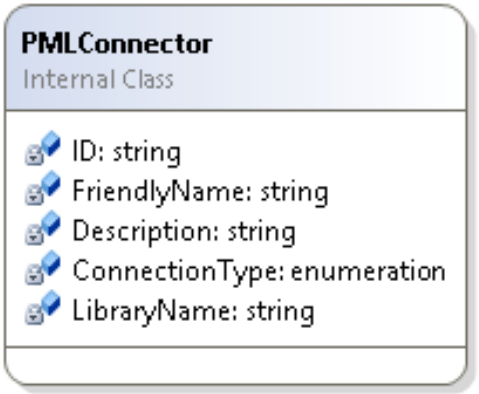

Figura 4.9: PMLConnector

O PMLConnector possui os seguintes atributos:

- ID – *String* que é a identificação única do elemento;
- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- ConnectionType – Enumeração que especifica o tipo de comunicação, como OPC UA, *Structured Query Language* (SQL), etc;
- LibraryName – *String* que indica o nome da biblioteca que deve ser usada na comunicação.

4.1.9 PMLEvent

Existem certos acontecimentos importantes num sistema de produção que precisam de atenção ou por parte do sistema em si, ou por parte do utilizador. A classe PMLEvent, representada na figura 4.10, abstrai este conceito e permite a definição de tipos concretos de eventos que vão ser especificados a seguir.

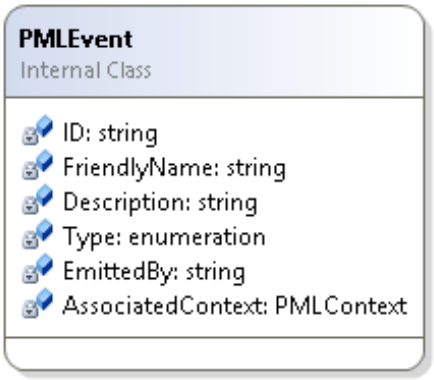

Figura 4.10: PMLEvent

Os atributos presentes nesta classe são:

- ID – *String* que é a identificação única do elemento;
- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- Type – Enumeração que especifica o tipo de evento;
- EmittedBy – *String* que referencia a fonte do evento;
- AssociatedContext – Elemento do tipo PMLContext que referencia em que contexto ocorreu determinado evento.

4.1.10 Tipos de Eventos

Existem diferentes eventos que podem ocorrer, estes vão estender a class PMLEvent e vão acrescentar informação relevante à sua definição. Os tipos de eventos possíveis podem incluir dados sobre falhas, sobre os *ramp-ups* dos subsistemas e componentes ou ainda informações sobre a recuperação de falhas. Na figura 4.11 é possível verificar todos estes detalhes.

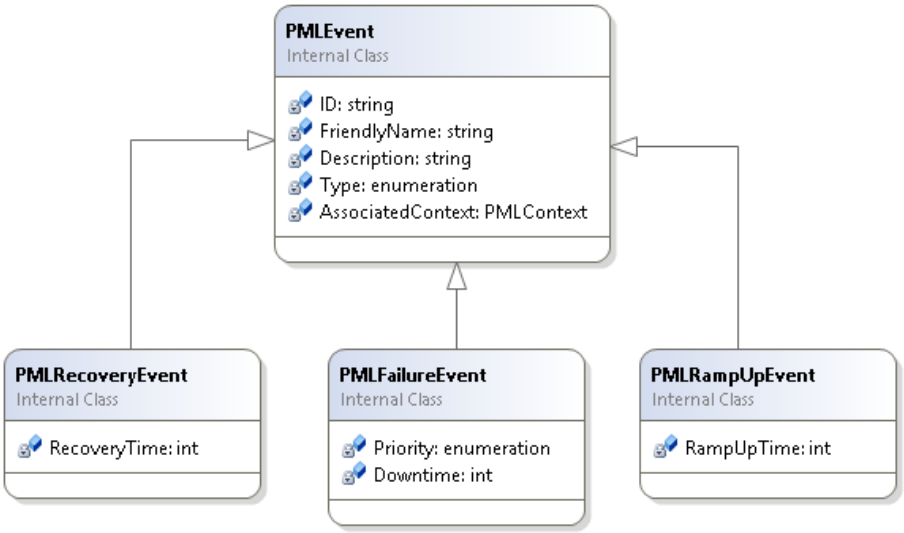

Figura 4.11: Tipos de Eventos

Além dos atributos herdados da classe PMLEvent temos para o conceito de PMLRecoveryEvent:

- RecoveryTime – Tempo necessário para a recuperação do sistema, desde um estado de falha a um estado normal de operação.

Para o conceito de PMLFailureEvent temos dois novos atributos, são eles:

- Priority – Enumeração que indica a prioridade de uma dada falha, por exemplo, baixa, média ou alta;
- Downtime – Tempo em que o sistema vai estar em baixo devido a uma determinada falha.

Por último temos para a classe PMLRampUpEvent:

- RampUpTime – Tempo necessário para fazer o *ramp-up* a um procedimento.

4.1.11 PMLContext

O conceito PMLContext vai providenciar em que contexto geral ocorreu um tipo de PMLEvent, como a localização e tempo de ocorrência. A descrição desta classe pode ser verificada na figura 4.12.

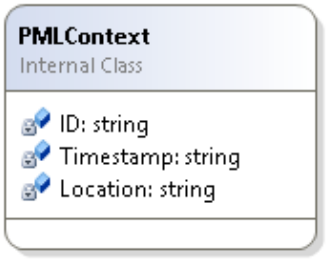

Figura 4.12: PMLContext

Esta classe possui os seguintes atributos:

- ID – *String* que é a identificação única do elemento;
- Timestamp – Indicador do instante em que algum acontecimento ocorreu;
- Location – *String* genérica que serve para indicar a localização onde alguma situação ocorreu, poderá ser um par de coordenadas ou um ID de um recurso ou subsistema.

4.1.12 PMLSystem

A classe PMLSystem é uma extensão da classe PMLSubsystem e funciona como um alto nível de abstração sendo possível assim abstrair a topologia inteira, os produtos e as simulações que possam ter sido executadas. Esta classe está descrita na figura 4.13.

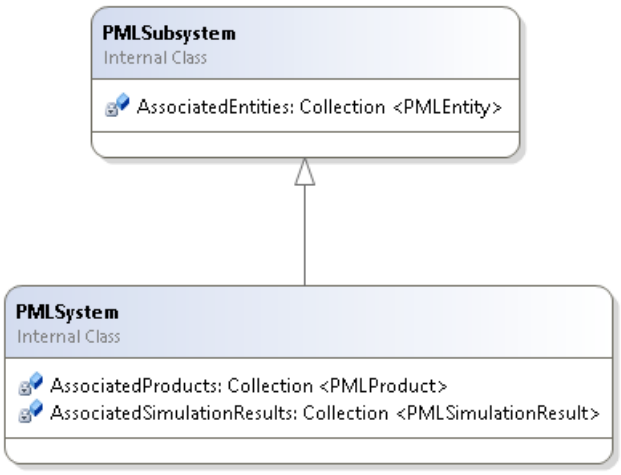

Figura 4.13: PMLSubsystem e PMLSystem

Como uma extensão do PMLSubsystem a classe PMLSystem vai herdar todos os seus atributos (que podem ser vistos na figura 4.4) e como se verifica na figura 4.13 vai possui mais dois, são eles:

- AssociatedProducts – Coleção de elementos da classe PMLProduct que representa os produtos associados a um determinado sistema;
- AssociatedSimulationResults – Coleção de elementos do tipo PMLSimulationResult que representa os resultados das simulações de um determinado sistema.

4.1.13 PMLSimulationResult

A classe PMLSimulationResult, descrita na figura 4.14, vai ser usada para representar um output de uma simulação, nomeadamente a lista de *Key Performance Indicators* ou KPI's para um determinado conjunto de configurações e/ou escalonamentos.

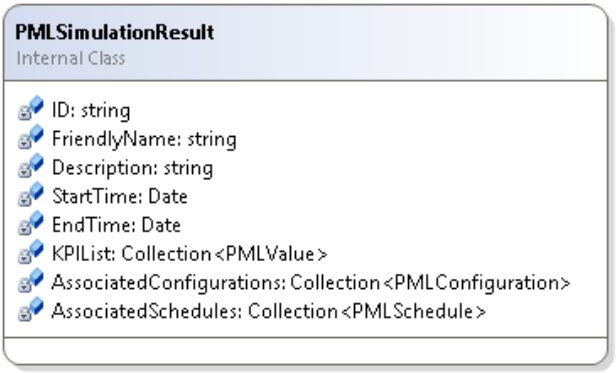

Figura 4.14: PMLSimulationResult

Os atributos presentes nesta classe são:

- ID – *String* que é a identificação única do elemento;
- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- StartTime – *Timestamp* em que a simulação começou;
- EndTime – *Timestamp* em que a simulação terminou;
- KPIList – Coleção de elementos do tipo PMLValue que representa os KPI's resultantes de uma dada simulação;
- AssociatedConfigurations – Coleção de elementos do tipo PMLConfiguration que indica as configurações do sistema associados a uma determinada simulação;

- AssociatedSchedules – Coleção de elementos do tipo PMLSchedule associados a uma determinada simulação.

4.1.14 PMLSchedule e PMLOperation

O PMLSchedule representa um escalonamento para uma determinada máquina, ou seja, contem todas as operações que a máquina precisa executar, a ordem da execução, a data de início e fim do agendamento. Já o PMLOperation descreve uma operação, isto é, a habilidade associada (atómica ou complexa), os requisitos de pessoal, as dependências da operação, a data em que a operação vai ser executada, entre outros. Estas duas classes encontram-se representadas na figura 4.15.

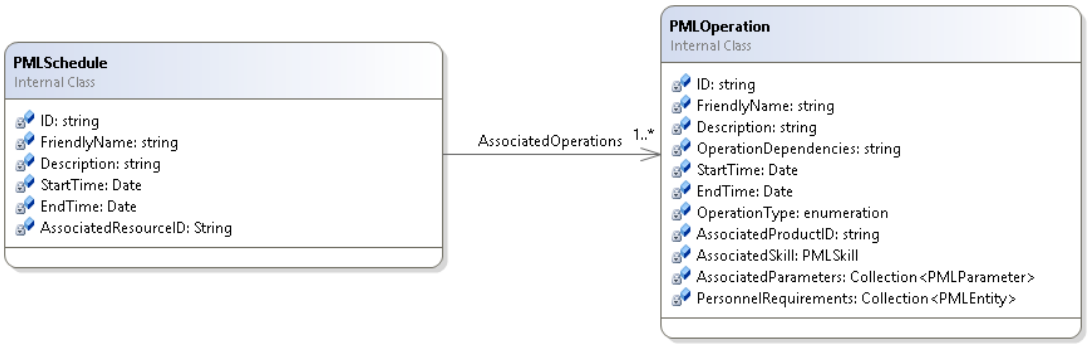

Figura 4.15: PMLSchedule e PMLOperation

O PMLSchedule tem os seguintes atributos:

- ID – *String* que é a identificação única do elemento;
- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- AssociatedResourceID – *String* que contém o ID do recurso associado a um dado Escalonamento;
- StartTime – *Timestamp* que indica a data e tempo mais cedo possível para um determinado Escalonamento começar;
- EndTime – *Timestamp* que indica a data e tempo mais tarde possível para um determinado Escalonamento acabar;
- AssociatedOperations – Coleção de elementos do tipo de PMLOperation que descrevem as operações e a ordem que serão executadas no recurso.

Já classe PMLOperation possui os seguintes atributos:

- ID – *String* que é a identificação única do elemento;

- FriendlyName – *String* que possui um nome legível do elemento;
- Description – *String* que contém uma descrição do elemento;
- OperationDependencies – *String* que contem a lista de todas as operações que precisam de estar completas para a execução de uma determinada operação;
- AssociatedProductID – *String* que contém o ID do produto associado a uma operação especifica;
- StartTime – *Timestamp* que indica a data em que uma operação especifica deve iniciar;
- EndTime – *Timestamp* que indica a data em que uma operação especifica deve terminar;
- OperationType – Enumeração que serve para identificar se a operação é realmente uma operação dita normal ou uma operação de manutenção;
- AssociatedSkill – Operação atómica ou complexa associada a uma determinada PMLOperation;
- AssociatedParameters – Parâmetros associados a uma operação específica;
- PersonnelRequirements – Lista de possíveis pessoas necessárias para a execução da operação.

Como se pode verificar ao longo do capítulo, o Modelo de Dados Comum foi desenhado com o proposito de ser o mais genérico possível, sendo que contém todas as classes necessárias para abstrair diversos tipos de sistemas, possivelmente com diferentes níveis de granularidade, o que garante a flexibilidade necessária no contexto da Indústria 4.0.

No próximo capítulo, vão ser apresentadas as várias tecnologias usadas na implementação do trabalho descrito neste documento, bem como a implementação das Interfaces Standard, do Modelo de Dados Comum e ainda de um middleware criado para posterior validação do trabalho realizado.

Capítulo 5

Implementação

Neste capítulo vai ser apresentada e descrita a implementação do Modelo de Dados Comum explicado no capítulo 4, das Interfaces Standard referidas no subcapítulo 5.2 e um middleware criado para posterior validação do trabalho realizado.

Para estas implementações foram usadas várias tecnologias sendo elas: Java, *Automation Markup Language* (AML), WinCC Open Architeture (WINCC OA) e Webservices baseados em REST.

A programação em Java foi escolhida pela sua fácil utilização e pela expriência adquirida ao longo do curso nesta linguagem, facilitando assim o trabalho de integração. Outro dos motivos que levou à sua escolha, foi a existência de uma camada de integração entre Java e WinCC OA. Igualmente, foi tido em consideração o facto do Java e do AutomationML terem ambos uma programação orientada a objetos.

A utilização do AutomationML no desenvolvimento do Modelo de Dados, teve em conta os resultados apresentados por Peres (Peres et al., 2016), que realizou um estudo com a finalidade de conhecer os vários standards existentes para modelação e representação de dados e as suas implementações. Este estudo foi realizado com o objetivo de encontrar o standard que mais se adapta ao projeto H2020 PERFoRM. Na secção 5.1 encontra-se a sua descrição sucinta.

Considerando a forte presença da Siemens nos Sistemas de Manufatura (com elementos como por exemplo os PLC's), foi escolhido o WinCC OA como parte do Middleware pois este conta com vários drivers, como OPC UA, já definidos, facilitando assim a integração com vários elementos presentes nos sistemas atuais da Indústria.

Como referido o Middleware foi desenvolvido usando o WinCC OA, mas também por outra tecnologia, o Apache CFX. Estas escolhas estão de acordo com as opções tomadas no H2020 PERFoRM.

5.1 Modelo de Dados

Como apresentado no subcapítulo 2.2 e referido em (Peres et al., 2016) foi realizado um estudo sobre softwares existentes para o desenvolvimento do modelo de dados. Foram analisadas sete ferramentas diferentes: IEC 61512 BatchML, IEC 62264 B2MML, ISO 15926 XMplant, IEC 62424 CAEX, IEC 62714 AutomationML, OPC UA's Data Model e MTConnect.

Estas ferramentas foram estudadas e analisadas segundo o suporte (total, parcial ou inexistente) de treze requisitos. A escolha destes requisitos teve em consideração os objetivos do H2020 PERFoRM e exemplo de alguns destes são: monitorização de qualidade, planeamento e escalonamento da produção, manutenção, controlo de processo entre outros. No estudo é possível verificar que nenhum dos standards cobre na totalidade todos os requisitos importantes.

Aliado ao suporte dos vários requisitos, considerou-se ainda neste estudo importância para os utilizadores finais de cada um dos critérios. Estes dois fatores juntos serviram para pontuar cada uma das tecnologias e concluir sobre as mesmas.

Segundo os resultados apresentados por Peres (Peres et al., 2016) nenhuma das tecnologias conseguiu pontuação final acima de 50% e as pontuações são bastante próximas umas das outras. A ferramenta que mais sobressaiu foi B2MML com uma pontuação de 35.7318% e em segundo lugar o AutomationML com 25.0641%.

Como referido, nenhuma das tecnologias abrange todos os requisitos necessários para o projeto PERFoRM e após uma análise mais profunda chegou-se à conclusão que apesar do AutomationML ter ficado em segundo lugar na classificação final, este podia ser facilmente estendido para cobrir os requisitos em falta, pois o B2MML apesar de maior pontuação parece ser menos flexível e ao acrescentar ou alterar definições deixaria de ser standard, e esta situação não acontece em AutomationML.

Assim sendo, o modelo de dados foi desenhado em AutomationML, tendo em conta todos os requisitos dos diferentes casos de estudo e das diferentes ferramentas software presentes no PERFoRM (PERFoRM Project, 2016b). O modelo final pode ser visto na figura 4.1 da secção 4.1.

5.1.1 AutomationML

Esta secção vai servir para introduzir o AutomationML e descrever um pouco as suas funcionalidades e o próprio editor.

O AutomationML é uma solução baseada em XML, aberta e independente do fabricante para trocas de dados focando-se maioritariamente em sistemas de engenharia de automação. Esta ferramenta é também capaz de cobrir toda a informação relevante em sistemas de engenharia de produção. A informação guardada em AutomationML baseia-se no paradigma de orientação a objetos. O AutomationML encontra-se inserido na família de standards IEC 62714 (INTERNATIONAL ELECTROTECHNICAL COMMISSION,

2014).

O AutomationML permite modelar quer objetos físicos quer lógicos e vários aspetos referentes aos mesmos. Os objetos modelados podem fazer parte de uma hierarquia, conter informações da sua descrição como geometria, cinemática ou até lógica, como informações lógicas de controlo.

O AutomationML integra diferentes formatos de dados baseados em XML, como se pode verificar na figura 5.1, sendo eles: CAEX, COLLADA e PLCopen. É através do CAEX (IEC 62424 (INTERNATIONAL ELECTROTECHNICAL COMMISSION, 2016)) que é possível descrever a topologia dos componentes, informações de rede e propriedades dos objetos como uma hierarquia no AutomationML. O CAEX também é a base para descrever a relação entre objetos em AML. A descrição geométrica e cinemática dos objetos é conseguida usando COLLADA 1.4.1 e 1.5.0 (ISSO/PAS 17506:2012). Para descrever a informação lógica de controlo o AutomationML usa PLCopen XML 2.0 e 2.0.1.

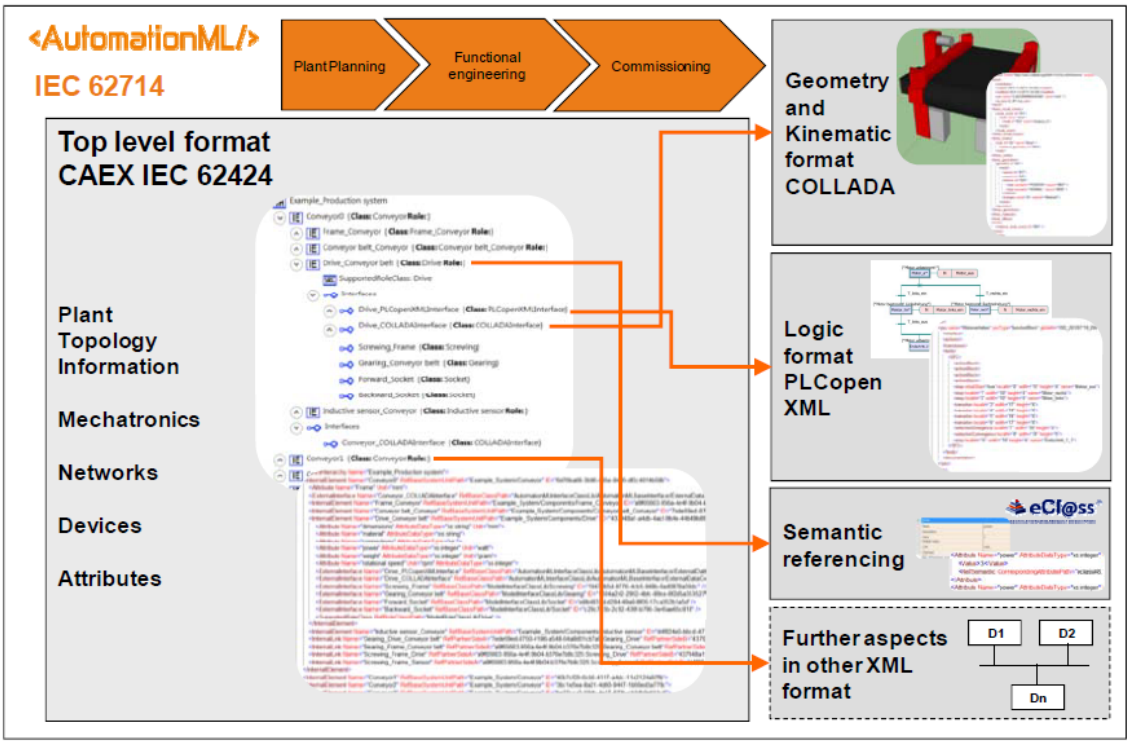

Figura 5.1: Constituição do AutomationML

Como referido, o AutomationML baseia-se no CAEX para a modelação da topologia e dos elementos do sistema, e oferece quatro formas de modelação. Estas formas de modelação estão representadas na figura 5.2.

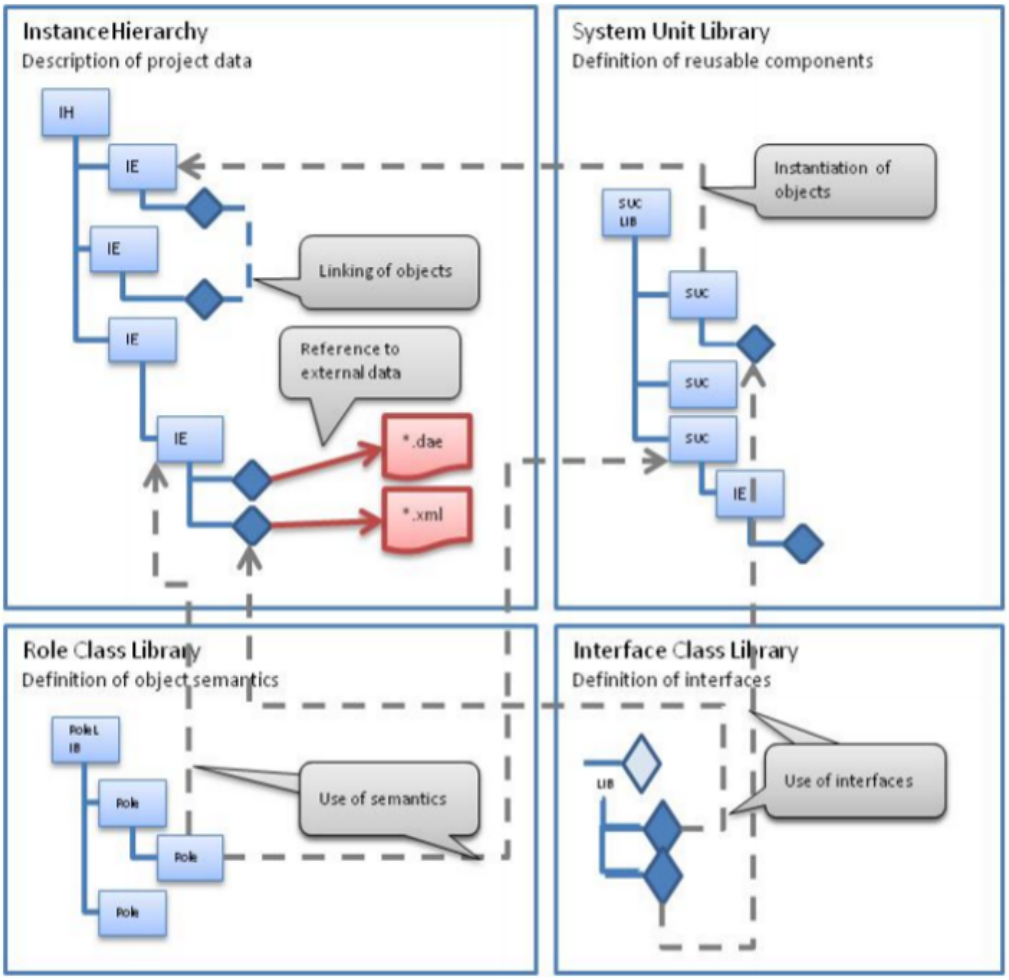

Figura 5.2: Formas de Modelação no AutomationML

Primeiramente temos as *Role Classes* que estão compreendidas dentro de uma biblioteca específica com o nome de *Role Class Library*. Uma *Role Class* é usada para descrever o contexto semântico de um objeto, sem qualquer pormenor da sua implementação técnica. No AutomationML já se encontram definidas algumas classes essenciais usadas normalmente em sistemas de engenharia, como por exemplo Recurso ou Produto. Todas as *Role Classes* têm de ter por base um role definido em AutomationML com o nome de AutomationMLBaseRole.

Seguidamente tem-se as Interfaces. Estas classes estão compreendidas dentro da biblioteca *Interface Class Library*. Estas classes definem as interfaces necessárias para relacionar elementos presentes em AutomationML e ficheiros ou outros elementos que não sejam base CAEX. Existe já interfaces definidas na biblioteca das Interfaces, como por exemplo a COLLADAInterface que é usada para relacionar um elemento a um ficheiro externo do tipo COLLADA.

Em terceiro, tem-se as *System Unit Classes* que vão ser usadas para definição dos componentes reutilizáveis do sistema ou como modelos (*templates* para a modelação de sistemas). Estas podem ser uma biblioteca de dispositivos e componentes dependentes

do vendedor ou por outro lado podem definir um conjunto de modelos usados para estruturar um Modelo de Dados de um sistema. Esta classe está compreendida na biblioteca *System Unit Class Library* e ao contrário das anteriores não existe qualquer classe deste tipo já definida no AutomationML.

Por fim, tem-se a *Instance Hierarchy* onde é instanciado o sistema que se deseja modelar. Esta é constituída por vários InternalElements que não são mais que os elementos que foram definidos na SystemUnitClassLibrary e que vão sendo reutilizados.

O Editor

Como se pode verificar na figura 5.3, o editor apresenta graficamente quatro divisões, sendo cada uma delas utilizada para cada uma das bibliotecas explicadas no subcapítulo anterior, são elas: RoleClassLib, SystemUnitClassLib, InterfaceClassLib e InstanceHierarchy.

O editor oferece ainda a possibilidade de visualização XML do elemento ou classe que estiver a ser definido, também visível na figura 5.3.

Existe do lado direito do editor uma coluna com vários separadores onde se definem diversas informações. Tem-se o cabeçalho (*header*) onde se definem informações como: Descrição, Versão ou alguma informação adicional de um determinado elemento. Nesta coluna tem-se ainda o separador dos atributos onde o utilizador pode definir os vários atributos pretendidos num elemento específico.

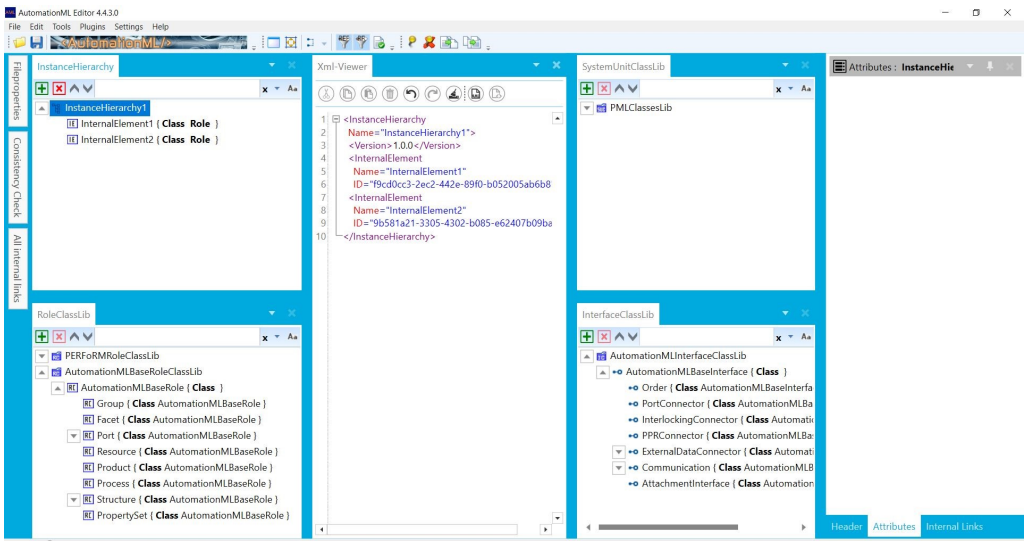

Figura 5.3: Editor do AutomationML

Depois desta breve introdução ao AutomationML e ao seu editor, vai ser apresentado o modelo de dados, descrito na secção 4.1, implementado neste software. Além dessa informação vão ser descritas algumas das classes, anteriormente explicadas, no editor com a sua descrição XML lado a lado.

5.1.2 Modelo de Dados

Na figura 5.4 e 5.5, encontra-se descrito o modelo de dados dentro do editor de AutomationML. Nestas figuras são apresentadas todas as *Role Classes* necessárias e as *System Unit Classes* que foram modeladas respetivamente.

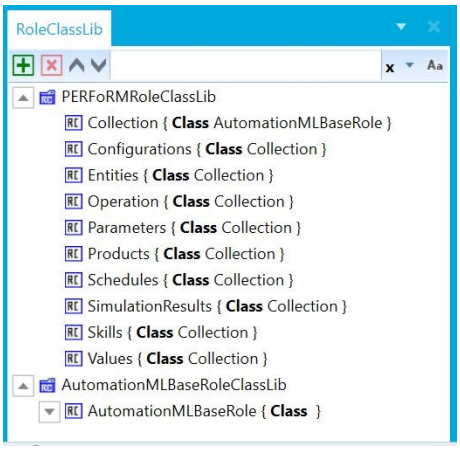

Figura 5.4: *Role Classes* definidas

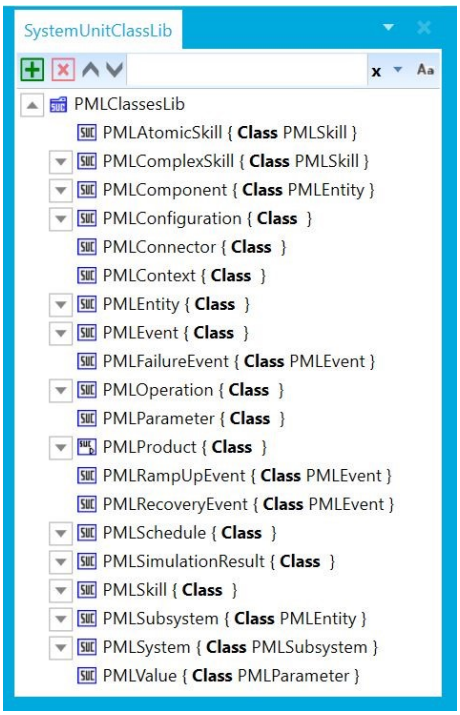

Figura 5.5: *System Unit Classes* definidas

Como se pode verificar na figura 5.4, todas as *Role Classes* definidas são extensões da *Role Classe Collection* que por sua vez estende a AutomationMLBaseRole. *A Role Classe Collection* foi definida por não existir nenhum *Role* já definido, dentro do AutomationML que sirva para o efeito.

Todas as *System Unit Classes* foram descritas, na secção 4.1, em detalhe com a explicação dos vários atributos. Neste subcapítulo vão ser apresentadas apenas algumas das classes presentes no modelo de dados e para cada uma delas vai ser apresentada uma breve descrição.

PMLEntity

A PMLEntity é a classe que representa de forma genérica os elementos presentes no *shop-floor*. Não vai ser diretamente usada, mas fornece toda a informação comum referente a componentes e subsistemas e permite ainda definir coleções genéricas de componentes e/ou subsistemas. Na figura 5.6 é apresentado o editor com foco nesta classe, mostrando o seu aspeto quer no editor quer em XML.

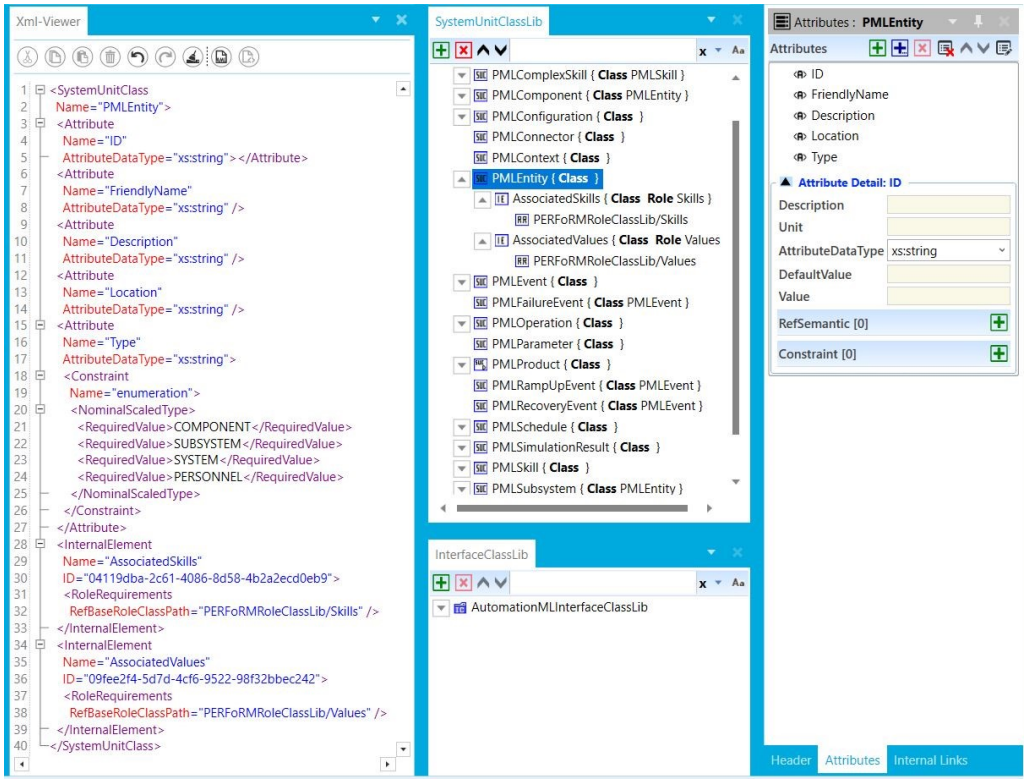

Figura 5.6: PMLEntity no editor de AutomationML

Como se pode verificar na figura 5.6, a classe PMLEntity tem dois elementos internos que são coleções de PMLSkills e de PMLValues (linhas 28 a 39 no XML-Viewer). Para definir a semântica associada a estas coleções, estes elementos internos possuem as Roles Classes respetivas, ou seja, a *Role Class Skills* e a *Role Class Values*.

PMLValue

A classe PMLValue permite a representação de informação relevante que é extraída de diferentes fontes, como por exemplo bases de dados ou PLC's. Estes dados dependem da exigência do consumidor, mas podem ser por exemplo valores referentes a temperaturas,

pressões ou um simples valor que funciona como uma *flag* (ligado ou desligado), entre tantos outros. Esta classe pode ser vista na figura 5.7.

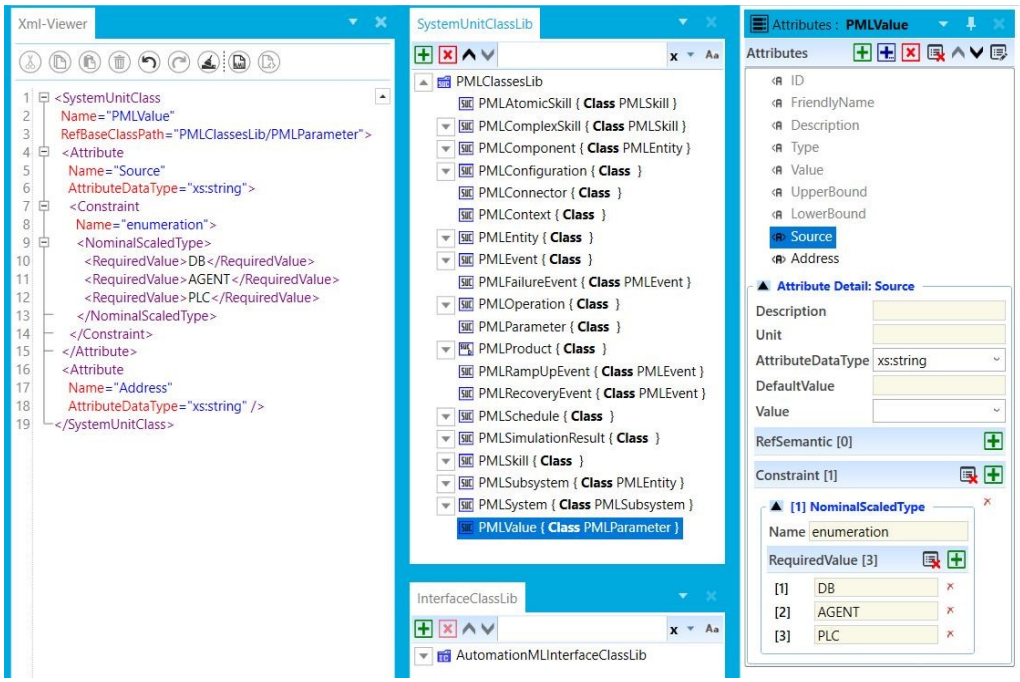

Figura 5.7: PMLValue no editor de AutomationML

Esta classe estende a classe PMLParameter logo possui todos os seus atributos. Este fenómeno pode ser verificado na figura 5.7 na janela de atributos, onde estão definidos a negrito os atributos referentes apenas à classe PMLValue que são o *Source* e o *Address*, enquanto todos os outros que estão "desvanecidos" são os atributos que pertencem à classe PMLParameter. No XML-Viewer podemos verificar que existe também a ausência desses mesmos atributos, pois não pertencem diretamente à classe.

É possível também verificar nas linhas 6 a 14 ou na janela dos atributos, na mesma figura, a forma utilizada para definir as enumerações. Estas enumerações são criadas quando existe algum atributo que tem um valor específico dentro de uma coleção, isto é, em casos onde só existam certas "constantes" ou opções possíveis para o valor de um determinado atributo, esta coleção de constantes esteja bem definida. Neste caso tem-se três constantes definidas, ou seja, o valor do *Source* só pode ser: DB, AGENT ou PLC.

PMLProduct

A classe PMLProduct abstrai um produto e as suas características. Possui uma descrição das habilidades necessárias para a sua produção e contém uma referência para um ficheiro COLLADA externo, que caso exista deverá conter toda a informação sobre as caraterísticas geométricas e cinéticas. Esta classe está apresentada na figura 5.8.

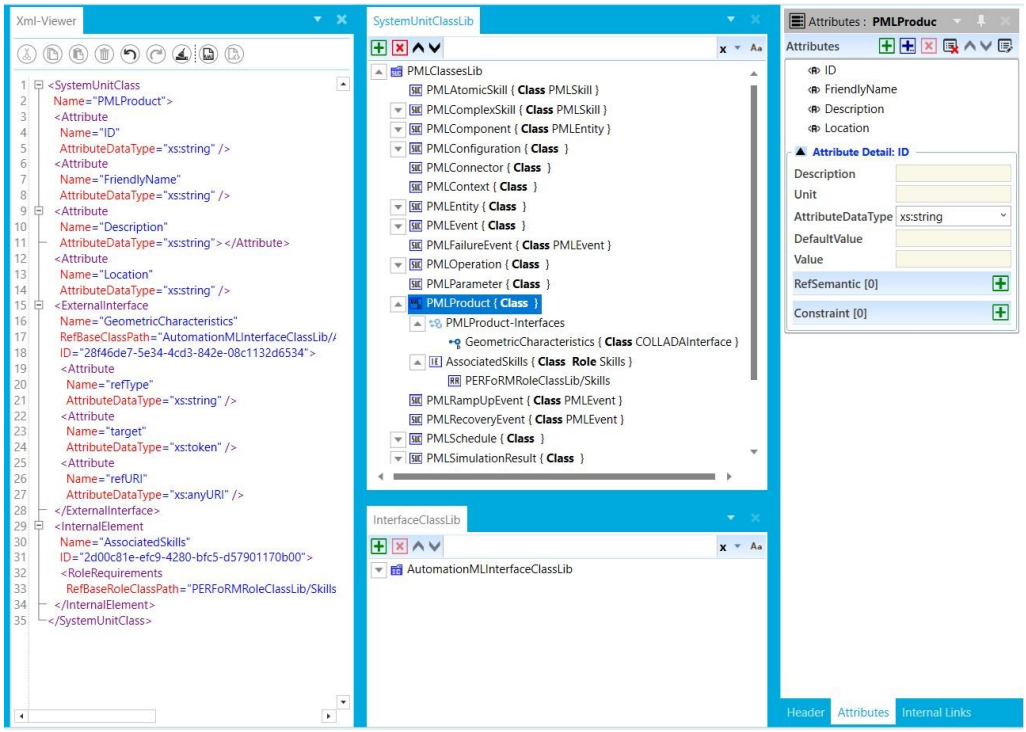

Figura 5.8: PMLProduct no editor de AutomationML

À semelhança dos outros exemplos, podemos ver os vários atributos presentes nesta classe quer no Xml-Viewer quer na janela de atributos. É possível ver também um elemento interno com o nome de AssociatedSkills que contém o *Role* que permite definir as coleções como acontecia na PMLEntity anteriormente referida.

Esta classe contém uma particularidade que não acontece com mais nenhuma classe neste Modelo de Dados. Esta particularidade é a existência da Interface para ficheiros externos, neste caso COLLADA. Esta interface pode ser vista na janela das SystemUnitClassLib.

5.2 Interfaces genéricas e standard

Como referido nos capítulos 2 e 3, as Interfaces Standard são um ponto chave para a interoperabilidade num contexto da Indústria 4.0 e consequentemente no H2020 PERFoRM. As interfaces standard são o núcleo para a plugabilidade e a interoperabilidade, permitindo a ligação entre hardware e software em camadas diferentes de manufatura de uma maneira transparente.

Foram definidas três Interfaces Standard diferentes. Tem-se uma Interface para as comunicações com as ferramentas software presentes no alto nível do sistema. Esta interface vai expor os métodos necessários para a ligação entre as ferramentas e o Middleware. Por outro lado, tem-se uma Interface que expõe os métodos necessários para o baixo nível (*shop-floor*) comunicar com o Middleware. Por último existe uma terceira interface que expõe os métodos necessários para a interação com o Modelo de Dados.

As várias Interfaces foram implementadas usando Java o que permite uma fácil integração com os restantes componentes do sistema.

5.2.1 Interface para o alto nível

Como descrito, esta interface vai permitir a ligação entre as ferramentas presentes no alto nível com o Middleware. Esta Interface está representada na figura 5.9.

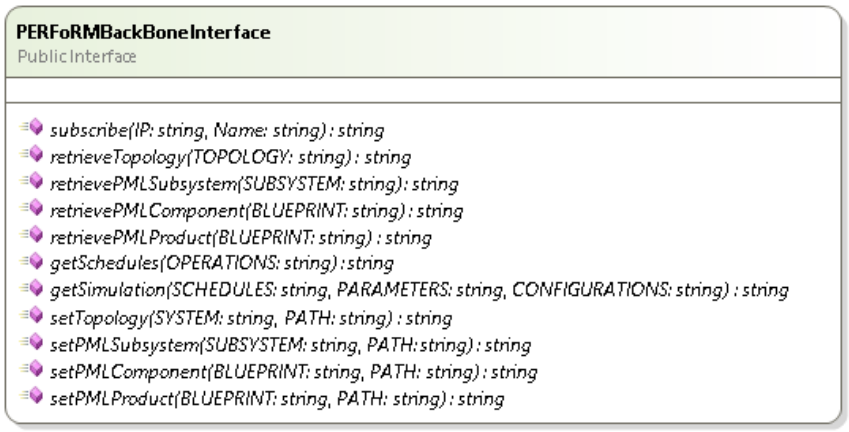

Figura 5.9: Interface de ligação com o alto nível

Como se pode ver, esta interface vai expor vários métodos necessários no alto nível do sistema. Os métodos vão ser descritos seguidamente:

- string Subscribe (string IP, string Name) – Este método é usado quando uma ferramenta é ligada no sistema, para dar conhecimento ao Middleware das habilidades que possui, para as disponibilizar para os restantes elementos do sistema.

Os restantes métodos podem ser divididos em duas categorias, ou são métodos de obtenção de dados ou pelo contrário, de escrita de dados. Os seguintes métodos são de obtenção de dados:

- string RetrieveTopology (string TOPOLOGY);
- string RetrieveSubsystem (string SUBSYSTEM);
- string RetrievePMLComponent (string BLUEPRINT);
- string RetrievePMLProduct (string BLUEPRINT);
- string getSchedules (string OPERATIONS);
- string getSimulation (string SCHEDULES, string PARAMETERS, string CONFIGURATIONS).

Estes métodos recebem alguma informação de *input*, no caso dos quatro primeiros métodos o *input* é o ID do produto desejado ou um caminho para o ficheiro da topologia do sistema por exemplo. No caso do getSchedules o *input* é a lista de operações que o Escalonador precisa de considerar para gerar escalonamentos. No método getSimulation, existem vários *inputs*, pois este necessita dos escalonamentos e/ou as configurações/parâmetros a ter em consideração. O retorno dos métodos são as classes PML correspondentes na forma de *string*, como por exemplo no retrieveTopology o retorno é um PMLSystem.

Já os restantes métodos são de escrita de dados, tem-se:

- string setTopology (string SYSTEM, string PATH);
- string setPMLSubsystem (string SUBSYSTEM, string PATH);
- string setPMLComponent (string BLUEPRINT, string PATH);
- string setPMLProduct (string BLUEPRINT, string PATH).

Estes métodos já têm dois *inputs*, primeiro os dados que se pretende escrever (uma topologia, um componente ou um produto por exemplo) e o caminho do ficheiro onde se pretende escrever os dados.

5.2.2 Interface para o baixo nível

A Interface para o baixo nível vai permitir uma ligação entre o middleware e os dispositivos presentes no *shop-floor*. Esta interface está representada na figura 5.10.

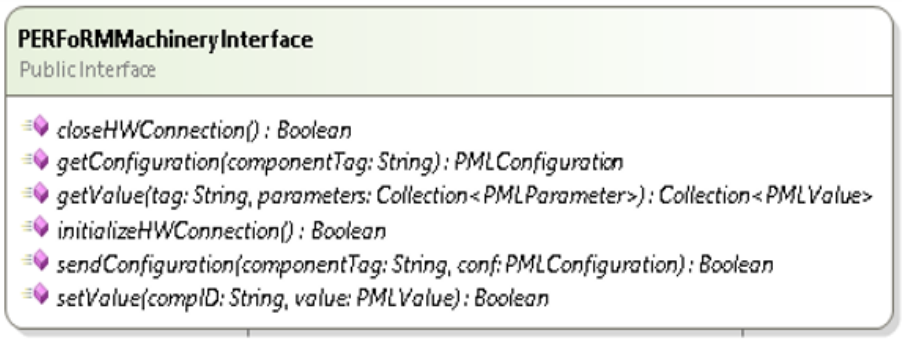

Figura 5.10: Interface para ligação com o baixo nível

Os métodos desta interface como acontece na anterior podem ser categorizados. Temos métodos de ligação aos dispositivos, de recolha de dados ou de envio de dados para os mesmos.

Os métodos de ligação aos dispositivos são:

- Boolean initializeHWConnection ();
- Boolean closeHWConnection ().

Estes dois métodos vão permitir inicializar ou fechar uma ligação a um dispositivo hardware presente no *shop-floor*. O retorno é um booleano, ou seja, *true* em caso de sucesso na operação ou *false* caso contrário.

Os métodos de leitura de dados dos dispositivos são:

- PMLConfiguration getConfiguration (string ComponentTag);
- Collection<PMLValue> getValue (string Tag, Collection <PMLParameter> parameters).

Estes métodos a partir de uma determinada *tag* ou ID vão ler a configuração ou um conjunto de valores de um determinado componente. Para a leitura de um conjunto de valores, é enviado como input um conjunto de parâmetros para especificar os valores a serem lidos, um exemplo é uma coleção com dois *timestamps* onde se pretende os valores lidos nesse intervalo de tempo.

Por fim tem-se os métodos de escrita, são eles:

- Boolean sendConfiguration (string ComponentTag, PMLConfiguration config);
- Boolean sendValue (string compID, PMLValue value).

Estes métodos ao contrário dos anteriores, vão servir para escrever uma determinada configuração ou valor num componente, onde o ID do mesmo é passado por parâmetro de entrada. Além do ID é passado por input a própria configuração (PMLConfiguration) ou o valor (PMLValue) que se pretende escrever.

5.2.3 Interface para o Modelo de Dados

A interface PERFoRMMLInterface foi criada para as interações entre o sistema e o modelo de dados. Esta interface vai expor as funcionalidades necessárias para obter ou escrever os dados necessários de um ficheiro aml. Esta interface está representada na figura 5.11.

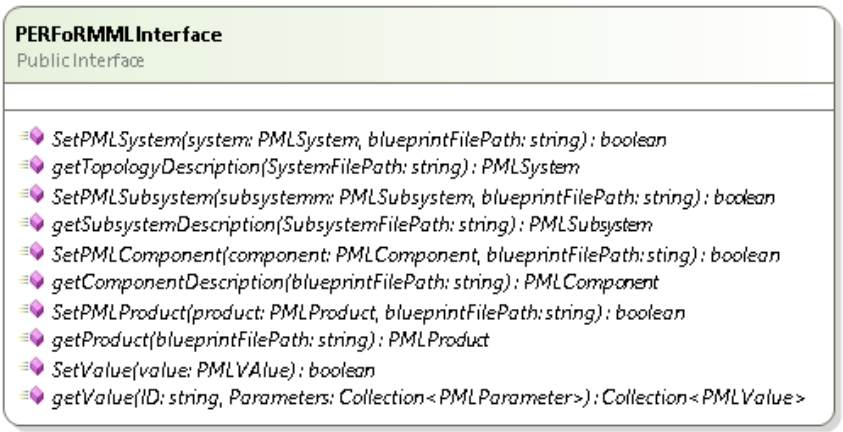

Figura 5.11: Interface para ligação ao Modelo de Dados

Como acontece com as Interfaces anteriormente apresentadas, é possível categorizar os vários métodos, na realidade os métodos existentes nesta Interface são de leitura ou escrita apenas.

Os métodos de leitura de dados são:

- PMLSystem getTopologyDescription (string SystemFilePath);
- PMLSubsystem getSubsystemDescription (string SubsystemFilePath);
- PMLComponent getComponentDescription (string BlueprintFilePath);
- PMLProduct getProduct (string BlueprintFilePath);
- PMLValue getValue (string ID, Collection <PMLParameter> Parameters).

Os quatro primeiros métodos têm apenas um parâmetro de entrada, sendo ele o caminho para o ficheiro de onde se pretende ler a topologia, um *blueprint* de um subsistema, componente ou produto. Já o getValue tem como *inputs* o ID do valor que se pretende ler, como uma lista de parâmetros que vai servir para especificar os valores que pretendem ser lidos, como um intervalo temporal por exemplo.

O retorno dos vários métodos de leitura são um elemento do tipo correspondente ao método, isto é, um elemento do tipo PMLSystem quando se lê a topologia por exemplo.

Os métodos de escrita são correspondentes aos métodos de leitura, ou seja:

- Boolean setPMLSystem (PMLSystem system, string blueprintFilePath);
- Boolean setPMLSubsystem (PMLSubsystem subsystem, string blueprintFilePath);
- Boolean setPMLComponent (PMLComponent componente, string blueprintFilePath);
- Boolean setPMLProduct (PMLProduct product, string blueprintFilePath);
- Boolean setPMLValue (PMLValue value).

Nos quatro primeiros métodos os parâmetros de entrada são o elemento que se pretende escrever (o sistema ou o *blueprint* de um produto por exemplo) e o caminho para o ficheiro a ser escrito. Já no setValue apenas o elemento é passado pois como não existe um blueprint para um valor, este vai ser procurado dentro da topologia e quando encontrado vai ser substituído pelo novo valor. O retorno destes métodos de escrita é um booleano, ou seja, *true* caso a operação tenha sido um sucesso ou *false* caso contrário.

5.3 AML Parser

Como anteriormente referido o modelo de dados foi criado em AutomationML, e como este tem como base CAEX, os ficheiros aml não podem ser lidos como um ficheiro normal

XML. Para ser possível fazer a conversão desse tipo de dados para um tipo de dados que fosse possível ler e escrever, foi necessário criar um *parser* para permitir a leitura e a escrita dos documentos aml. Este *parser* foi desenvolvido usando a linguagem de programação Java, pois como explicado no inicio do capítulo existe uma boa ligação entre AutomationML e Java por serem ambos orientados a objetos. O Java como anteriormente referido conjuga bem com os restantes elementos do sistema.

Para ser possível a leitura dos ficheiros aml, foram extraídas as classes existentes no esquema XML do CAEX usando a biblioteca *Java Architecture for XML Binding* (JAXB). As classes extraídas estão representadas na figura 5.12.

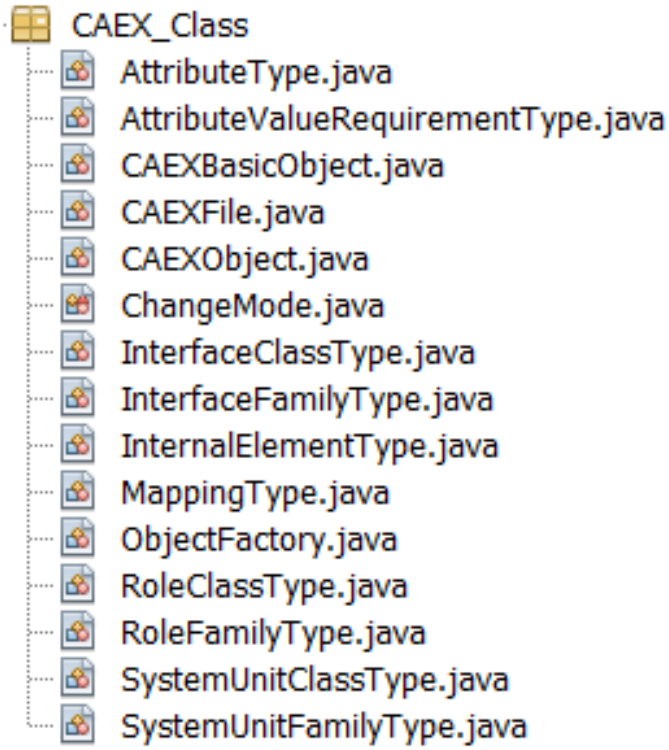

Figura 5.12: Classes extraídas do esquema XML do CAEX

Estas classes vão permitir assim que os ficheiros aml sejam corretamente lidos e convertidos de aml para um objeto do tipo CAEXFile. Este objeto vai assim conter todas as informações presentes no ficheiro aml, como as SystemUnitClasses por exemplo. Depois desta conversão já é possível trabalhar em Java e obter todas as informações necessárias. Nas próximas subsecções vai ser descrito o modelo de dados, a leitura e escrita de ficheiros AML.

5.3.1 Modelo de Dados

Para o *parser*, foi necessário criar uma classe Java para cada classe definida no modelo de dados, descrito na secção 4.1. Estas classes Java podem ser vistas na figura 5.13.

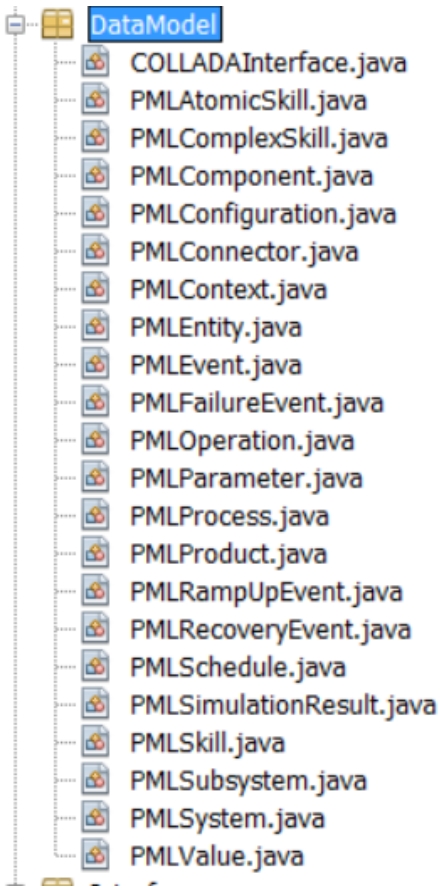

Figura 5.13: Classes Java do Modelo de Dados

Estas classes têm definidos todos os atributos anteriormente descritos e possuem todos os métodos necessários (*setters* e *getters*) para obter e escrever os mesmos. Cada classe possui ainda vários tipos de construtores para quando existe a necessidade de instanciar uma determinada classe com ou sem parâmetros.

5.3.2 Leitura de ficheiro aml

Os ficheiros aml têm por base os ficheiros XML. Assim utilizou-se a biblioteca JAXB (biblioteca que permite com base em certas classes extrair a informação de um ficheiro do tipo XML) em conjunto com as classes extraídas anteriormente do CAEX para obter as informações existentes no ficheiro aml. Para extrair esta informação foi usado o método *unmarshal*.

Tendo a extração de dados dos ficheiros aml e o modelo de dados definido é possível então criar o *parser*. O *parser* vai ser uma classe chamada AMLParser, e esta possui três variáveis, que se podem ver na fig 5.14.

```
public class AMLParser {

    private File file;
    private String FilePath;
    private AMLUnmarshaller u;
```

Figura 5.14: Variáveis existentes na classe AMLParser

A *string FilePath* vai indicar o caminho para o ficheiro aml que queremos extrair informação. O *File file* é o ficheiro em Java que vai ser criado a partir do *FilePath*. Por fim, *u* do tipo AMLUnmarshaller (classe criada especificamente para fazer *unmarshal* a um ficheiro aml) é o unmarshal que vai ser usado para ler os dados.

Métodos

A classe AMLParser vai expor vários métodos, todos eles usam o *file* criado a partir do *FilePath* anteriormente referido. Um ponto que se deve considerar é a existência de ficheiros aml que são *Blueprints*, ou seja, há ficheiros aml que devem possuir apenas um elemento de um determinado tipo (componente ou produto) que vão servir para casos em que tenhamos por exemplo vários componentes idênticos (robots) que vão divergir apenas no ID. Os métodos públicos são:

- public PMLSystem LoadTopology () – Método que extrai a informação do ficheiro aml e retorna um objeto do tipo PMLSystem que contém a informação da topologia do sistema caso exista senão é retornado *null*;
- public PMLSubsystem LoadSubsystem () – Método que extrai a informação do ficheiro aml e retorna um objeto do tipo PMLSubsystem que contém a informação sobre esse subsistema. Neste método é extraída a informação referente ao primeiro subsistema que aparece no ficheiro aml caso exista, senão é retornado *null*;
- public PMLSubsystem LoadSubsystem (String ID) – Método que extrai a informação do ficheiro aml e procura um subsistema com o ID passado em parâmetro. Caso exista é devolvido um objeto do tipo PMLSubsystem com a informação referente a esse subsistema, senão é retornado *null*;
- public PMLComponent LoadComponent () – Método que extrai a informação do ficheiro aml e retorna um objeto do tipo PMLComponent que contém a informação sobre esse componente. Neste método é extraída a informação referente ao primeiro componente que aparece no ficheiro aml caso exista, senão é retornado *null*;
- public PMLComponent LoadComponent (String ID) – Método que extrai a informação do ficheiro aml e procura um componente com o ID passado em parâmetro. Caso exista é devolvido um objeto do tipo PMLComponent com a informação referente a esse componente, senão é retornado *null*;

- public PMLProduct LoadProduct () – Método que extrai a informação do ficheiro aml e retorna um objeto do tipo PMLProduct que contém a informação sobre esse produto. Neste método é extraída a informação referente ao primeiro produto que aparece no ficheiro aml caso exista, senão é retornado *null*;

- public PMLProduct LoadProduct (String ID) – Método que extrai a informação do ficheiro aml e procura um produto com o ID passado em parâmetro. Caso exista é devolvido um objeto do tipo PMLProduct com a informação referente a esse produto, senão é retornado *null*;

A classe possui outros métodos internos que auxiliam a programação dos métodos públicos anteriormente descritos. Além destes possui ainda os *Getters* e *Setters* para as variáveis existentes.

5.3.3 Escrita de ficheiro aml

Para a escrita de um ficheiro aml vai ser usada a mesma biblioteca, ou seja, JAXB mas desta vez o método usado é o *marshal*. Como aconteceu na leitura, foi criada uma classe especifica para usar o método *marshal* e escrever um ficheiro aml, com o nome de AML-Marshaller.

Foi criada uma classe com o nome de AMLWriter para lidar com a escrita. Esta classe possui quatro variáveis (figura 5.15):

```
public class AMLWriter {

    private String FilePath;
    private String PERFoRMMLFilePath;
    private File AMLFile;
    private CAEXFile PERFoRMML;
```

Figura 5.15: Variáveis existentes na classe AMLWriter

A *string FilePath* indica o caminho e o nome do ficheiro desejado para o ficheiro aml que será escrito, já a *string PERFoRMMLFilePath* serve para indicar a localização do ficheiro aml com o modelo de dados definido para ser possível transportar toda essa informação para o novo ficheiro aml. O *File AMLFile* é um objeto do tipo *File* criado a partir do caminho indicado pelo *FilePath*, e por fim o *CAEXFile* é o objeto que tem definido o modelo de dados.

Métodos

Esta classe possui diferentes versões para o construtor, estas podem ser vistas na figura 5.16.

```
public AMLWriter() {...6 lines }

public AMLWriter(String FilePath) {...6 lines }

public AMLWriter(String FilePath, String PERFoRMMLFilePath) {...6 lines }

public AMLWriter(String FilePath, CAEXFile PERFoRMML) {...6 lines }

public AMLWriter(CAEXFile PERFoRMML) {...6 lines }
```

Figura 5.16: Métodos construtores para a classe AMLWriter

Além dos métodos construtores esta classe disponibiliza outros métodos que são:

- public boolean writeAMLSystem (PMLSystem system) – Método que recebe por parâmetro um objeto do tipo PMLSystem que contém toda a informação de um determinado sistema e escreve um ficheiro aml com essa mesma informação. Caso a escrita seja efetuada sem problemas retorna *TRUE*, caso contrário o retorno é *FALSE*;

- public boolean writeAMLSubsystem (PMLSubsystem subsystem) – Método que recebe por parâmetro um objeto do tipo PMLSubsystem que contém toda a informação sobre um determinado subsistema e escreve um ficheiro aml com essa mesma informação. Caso a escrita seja efetuada sem problemas retorna *TRUE*, caso contrário o retorno é *FALSE*;

- public boolean writeAMLComponent (PMLComponent component) – Método que recebe por parâmetro um objeto do tipo PMLComponent que contém toda a informação sobre um determinado componente e escreve um ficheiro aml com essa mesma informação. Caso a escrita seja efetuada sem problemas retorna *TRUE*, caso contrário o retorno é *FALSE*;

- public boolean writeAMLProduct (PMLProduct product) – Método que recebe por parâmetro um objeto do tipo PMLProduct que contém toda a informação sobre um determinado produto e escreve um ficheiro aml com essa mesma informação. Caso a escrita seja efetuada sem problemas retorna *TRUE*, caso contrário o retorno é *FALSE*;

- public boolean writeAMLSchedule (PMLSchedule schedule) – Método que recebe por parâmetro um objeto do tipo PMLSchedule que contém toda a informação sobre um determinado escalonamento e escreve um ficheiro aml com essa mesma informação. Caso a escrita seja efetuada sem problemas retorna *TRUE*, caso contrário o retorno é *FALSE*;

- public Boolean writeAMLSimulation (PMLSimulationResult simulation) – Método que recebe por parâmetro um objeto do tipo PMLSimulationResult que contém toda a informação sobre uma determinada simulação e escreve um ficheiro aml com essa

mesma informação. Caso a escrita seja efetuada sem problemas retorna *TRUE*, caso contrário o retorno é *FALSE*.

Esta classe também possui outros métodos privados que servem de auxiliares aos métodos públicos. Além desses, expõe os *Setters* e o *Getters* para as variáveis anteriormente referidas.

5.4 Middleware

A camada de Middleware, como anteriormente referido, é bastante importante no contexto da Indústria 4.0 e consequentemente no projeto H2020 PERFoRM. Apesar do foco do trabalho desenvolvido ser o Modelo de Dados Comum e as Interfaces Standard e não esta camada, de forma a validar o trabalho, foi criado um Middleware para se obter um sistema completo.

Como introduzido no inicio do capítulo, o Middleware vai ser constituído por duas tecnologias diferentes. De maneira ao trabalho desenvolvido manter-se completamente alinhado ao projeto H2020 PERFoRM, as tecnologias escolhidas para desenvolver a camada de Middleware foram o CXF Apache (Apache Software Foundation, 2017) e o WINCC OA.

Por um lado, tem-se Apache CXF, uma plataforma aberta de serviços. Esta plataforma vai habilitar a implementação dos serviços necessários. Vários protocolos são suportados por esses serviços como SOAP, XML/HTTP e RESTful HTTP. Esta plataforma tem uma performance bastante boa, é reconfigurável e intuitiva. O CXF foi desenhado para oferecer uma arquitetura que permite plugabilidade e para suportar não só XML, mas também por exemplo *JavaScript Object Notation* (JSON), independentemente do tipo de transporte de dados.

Em combinação com o Apache CXF foi escolhido o WinCC OA. É um sistema SCADA (*Supervisory Control and Data Acquisition*) orientado para objetos, ou seja, foi desenhado para aquisição e exibição de dados. Seguindo uma abordagem aberta, permite facilmente integrar vários componentes ou tecnologias através da criação de drivers individuais. Uma das vantagens é a existência de bastantes drivers já definidos, como por exemplo OPC UA.

Na secção 5.4.1 e na secção 5.4.2 vai ser descrito em promenor a implementação dos serviços em Apache CXF e a implementação em WinCC OA respetivamente.

5.4.1 Serviços

Como referido, a escolha para a implementação dos serviços foi a tecnologia Apache CXF, pois além de ser intuitiva e de fácil utilização é uma das tecnologias presentes no projeto H2020 PERFoRM. Os serviços vão ser desenvolvidos usando REST.

De forma a existir uma troca de dados standard entre todos os elementos do sistema, foi definido no projeto que as mensagens trocadas terão de ser JSON. O JSON deriva de

JavaScript (o que o faz ser bastante usado) e é um formato de dados aberto, independe da linguagem e legível ao humano. Este formato consiste em pares de atributos-valor, um exemplo pode ser visto na figura 5.17.

Figura 5.17: Resultado de um serviço em JSON

Como se pode verificar na figura, JSON é essencialmente texto, ou seja, uma *string* o que torna as trocas de dados bastante simples, de fácil e rápido processamento. A *string* vai conter toda a informação que se deseja transmitir. Para escrever e ler JSON é usada a biblioteca JACKSON e as classes e os métodos disponibilizados por esta, neste caso é usada a classe *Mapper* que vai servir para transformar a informação em *string* e transformar de string de volta ao seu estado original. Para estas transformações são usados os métodos *writeAsString* e *readFromString*. Assim todos os retornos e parâmetros dos serviços vão ser strings.

Os serviços que o middleware vai disponibilizar são:

- public String subscribe (String IP, string Name) – Serviço que permite às aplicações o seu registo no middleware, de forma a disponibilizar as suas funções como serviços;
- public String disconnect (string Name) – Serviço que é usado quando uma aplicação se desliga do Sistema, para actualizar a lista de ferramentas ligadas ao mesmo;
- public String retriveTopology (String FilePath) – Serviço que retorna uma *string* que contém a informação toda da topologia do sistema. É passado por parâmetro o nome/caminho do ficheiro desejado.
- public String retriveSubsystem (String FilePath, String ID) – Serviço que retorna uma *string* que contém a informação sobre um determinado subsistema. É passado o nome/caminho do ficheiro desejado e o ID do subsistema pretendido;

- public String retriveComponent (String FilePath, String ID) – Serviço que retorna uma *string* que contém a informação sobre um determinado componente. É passado por parâmetro o nome/caminho do ficheiro desejado e o ID do componente pretendido;

- public String retrieveProduct (String FilePath, String ID) – Serviço que retorna uma *string* que contém a informação sobre um determinado produto. É passado por parâmetro o nome/caminho do ficheiro desejado e o ID do produto pretendido;

- public String getSchedules (String operations) – Serviço que vai retornar uma coleção com elementos do tipo PMLSchedule, na forma de *string*, resultantes de um pedido de escalonamentos ao Escalonador. É passado por parâmetro uma coleção com elementos do tipo PMLOperation, como *string*, necessários por esta ferramenta para gerar os escalonamentos;

- public String getSimulation (String schedules, String configurations, String parameters) – A ferramenta de simulação vai necessitar de informações como os escalonamentos, configurações ou parâmetros de simulação, por isso esta informação é passada por parâmetros de entrada. O resultado é uma *string* que contém informação sobre uma simulação realizada;

- public String setTopology (String topology, String FilePath) – Serviço que permite a escrita de um ficheiro com a topologia de um determinado Sistema. Tanto a informação da topologia como do ficheiro a ser escrito é passado por parâmetro;

- public String setSubsystem (String subsystem, String FilePath) – Serviço que permite a escrita de um ficheiro com a informação de um determinado subsistema. Tanto a informação do subsistema como do ficheiro a ser escrito é passado por parâmetro;

- public String setComponent (String component, String FilePath) – Serviço que permite a escrita de um ficheiro com a informação de um determinado componente. Tanto a informação do componente como do ficheiro a ser escrito é passado por parâmetro;

- public String setProduct (String product, String FilePath) – Serviço que permite a escrita de um ficheiro com a informação de um determinado produto. Tanto a informação do produto como do ficheiro a ser escrito é passado por parâmetro;

- public String setSchedules (String schedules, String FilePath) – Serviço que permite a escrita de um ficheiro com a informação de determinados escalonamentos. Tanto a informação dos escalonamento como do ficheiro a ser escrito é passado por parâmetro;

- public String setSimulation (String Simulation, String FilePath) – Serviço que permite a escrita de um ficheiro com a informação de uma determinada simulação. Tanto a informação da simulação como do ficheiro a ser escrito é passado por parâmetro.

5.4.2 WinCC Open Architeture

O WinCC OA (ETM, s.d.-b) faz parte integrante do middleware devido à sua facilidade de ligação ao hardware existente em ambientes reais de manufatura. Esta facilidade existe, pois o software possui imensos drivers já definidos para ligação com os protocolos mais comuns na Indústria, como por exemplo, driver para OPC UA, *Open Platform Communications Data Access* (OPC DA), *Transmission Control Protocol/Internet Protocol* (TCP/IP) e SIMATIC S7. Além dos drivers definidos permite ainda ligações externas através de uma API C++.

O WinCC OA conta com funcionalidade HMI (*Human Machine Interaction*), alarmes, gráficos e tendências ou mesmo armazenamento de dados históricos. As funcionalidades de HMI foram usadas na validação dos resultados, permitindo assim uma visualização do está a ocorrer no sistema. Esta HMI pode ser vista na secção 6.2.

No WinCC OA vão ser definidos vários DataPoints e os respetivos nós necessários para um determinado sistema. Os DataPoints podem ser vistos como uma classe e os nós como os atributos dessa mesma classe. No capitulo 6 encontra-se um exemplo dessa definição, com um DataPoint definido e vários nós (secção 6.2). Na figura 5.18 está demonstrada uma interação entre uma aplicação genérica, o Middleware e o Modelo de Dados.

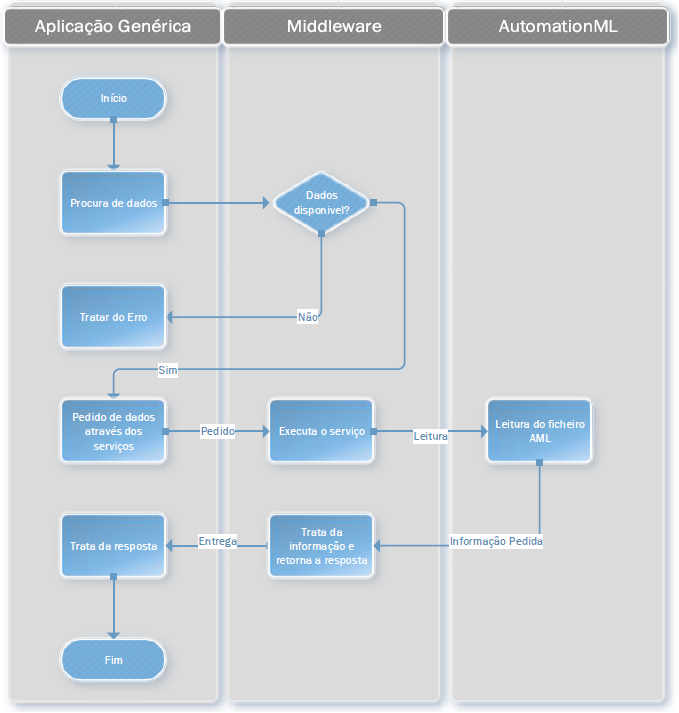

Figura 5.18: Interação entre um Aplicação Genérica, o Middleware e o Modelo de Dados

Como se pode ver na figura 5.18, a aplicação vai perguntar ao Middleware se existem informações sobre determinado elemento, valor, etc. Este por sua vez tem dois resultados possíveis, ou responde afirmativo caso a informação desejada esteja disponível ou responde negativo caso contrário. Caso não exista informação a ferramenta vai tratar do erro, mas no caso de existir, esta vai então pedir essa informação. Esse pedido passa pelo Middleware que vai executar um pedido de leitura ao Modelo de Dados. Após receber os dados pedidos o Middleware vai processar a informação e reencaminhá-la para a aplicação.

No próximo capítulo, será apresentado um caso de estudo (ambiente simulado) que irá servir para validar as Interfaces, o Modelo de Dados e o Middleware. Estes elementos vão ser testados em duas aplicações software.

Capítulo 6

Resultados e Validação

Como anteriormente referido, o trabalho desenvolvido enquadra-se no projeto H2020 PERFoRM que possui quatro casos de estudo diferentes. Como os casos de estudo ainda estão em fase de preparação não foi possível validar o Modelo de Dados e as Interfaces Standard nos mesmos. Assim sendo foi necessário utilizar um ambiente simulado para confirmar os resultados e poder validar os mesmos.

Utilizou-se um ambiente simulado em V-REP (Peres, Rocha & Barata, 2017) para simular o *shop-floor* de uma linha de produção. Este ambiente consiste numa linha de produção em que os produtos sofrem um processo de soldadura.

Numa vertente de operador e para teste direto da implementação, utilizou-se as funcionalidades do WinCC OA para criar uma interface gráfica (HMI) 2D que representa completamente o ambiente simulado apresentado em (Peres et al., 2017).

No âmbito do projeto H2020 PERFoRM, foram desenvolvidas duas ferramentas software dentro do grupo *Robotics & Industrial Complex Systems* (RICS), e estas foram usadas para a validação do trabalho desenvolvido. Uma destas ferramentas é o Escalonador que através de informações do sistema, e das operações desejadas gere escalonamentos considerando várias situações diferentes para um determinado sistema de produção. A segunda ferramenta que foi usada para validação dos resultados é uma ferramenta de análise de dados em tempo real.

Apesar destas ferramentas não fazerem parte do trabalho desenvolvido, como são parte integrante do sistema como um todo, nas próximas secções além dos resultados e validações vai ser apresentada uma breve descrição das mesmas.

6.1 Ambiente Simulado

Como referido no início do capítulo, existe um ambiente simulado criado em V-REP que representa uma linha de produção e é constituída por dois tapetes rolantes (para transporte das peças) e dois robots. Cada um dos robots tem a sua funcionalidade, um é responsável pela soldadura enquanto outro é de transporte, ou seja, pega nas peças e transporta-as para outro lado. Este ambiente está representado na figura 6.1 (Peres et al., 2017).

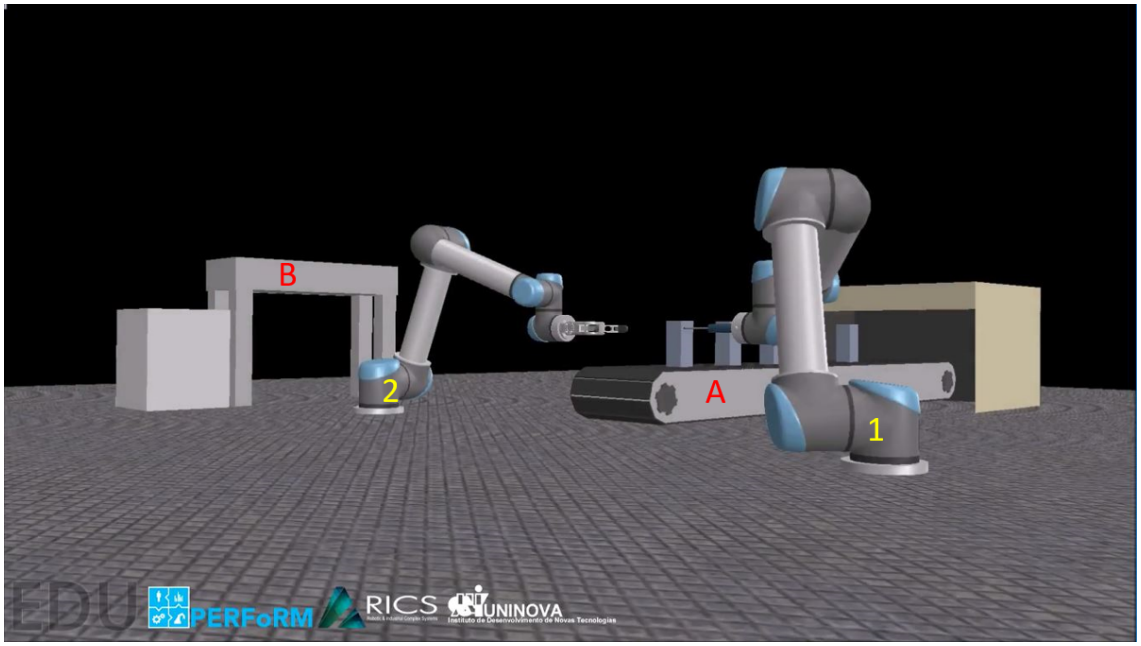

Figura 6.1: Ambiente Simulado em V-REP

O funcionamento desta linha de produção é o seguinte:

- Entrada da peça na linha pelo primeiro tapete rolante (tapete A), seguindo caminho até à estação onde vai sofrer um processo de soldadura;
- Aquando a chegada da peça o robot de soldadura (robot 1) vai trabalhar a peça (soldar);
- A peça ao estar pronta, é agarrada pelo segundo robot (robot 2) e transportada até ao segundo tapete (tapete B);
- Já no segundo tapete rolante a peça percorre o seu caminho até um depósito.

Como referido anteriormente, foi usado o WinCC OA como parte da solução de middleware, mas foi também usado para criar uma HMI que representa em 2D o ambiente referido neste subcapítulo. Este ambiente 2D vai ser apresentado na próxima secção.

6.2 WinCC OA HMI

Considerando a vertente de operador, ter um ambiente gráfico com todas as informações necessárias sobre o que está a acontecer no *shop-floor* é uma mais valia, pois a máquina que está a correr o software poderá estar longe da zona de maquinaria. Assim é possível mesmo remotamente visualizar o que está a acontecer na linha de produção.

Assim sendo, utilizou-se as funcionalidades do WinCC OA para a criação de um HMI que representa por completo o ambiente simulado anteriormente apresentado. Como o Middleware encontra-se numa máquina diferente à que corre o ambiente simulado, esta visualização permite graficamente verificar que está a acontecer no V-REP.

Este HMI pode ser visto na figura 6.2.

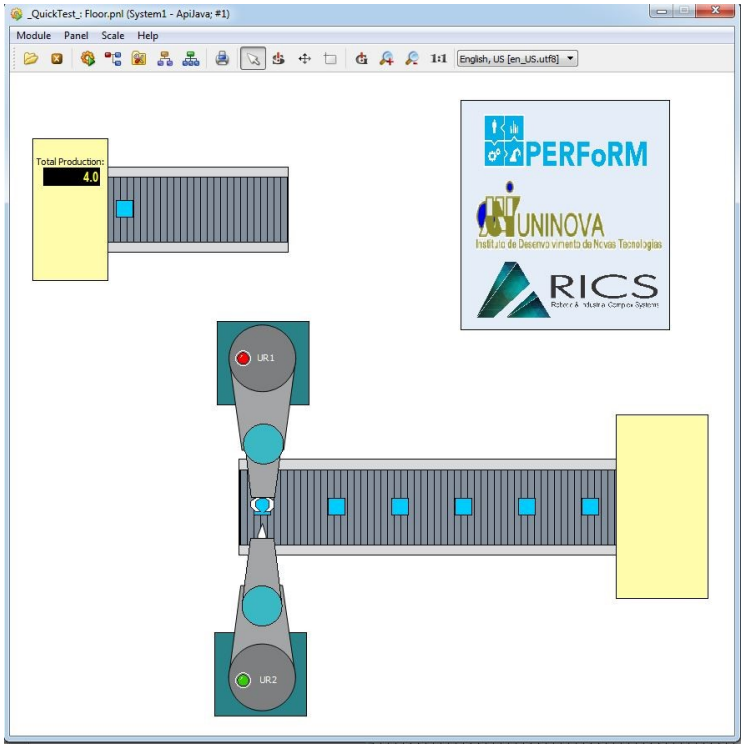

Figura 6.2: Ambiente Simulado em 2D

Como se pode ver, tem-se em 2D todos os elementos presentes no ambiente simulado. Os vários elementos presentes vão mover-se à medida que os dados provenientes do ambiente simulado vão chegando, como por exemplo o ângulo em que o robot UR1 (robot de *pick and place*) se encontra.

Neste sistema apenas foi necessário criar um *DataPoint* (ou DP). Este é constituído por vários nós, sendo atualizados quando existe alguma alteração no ambiente simulado. Este DP está representado na figura 6.3.

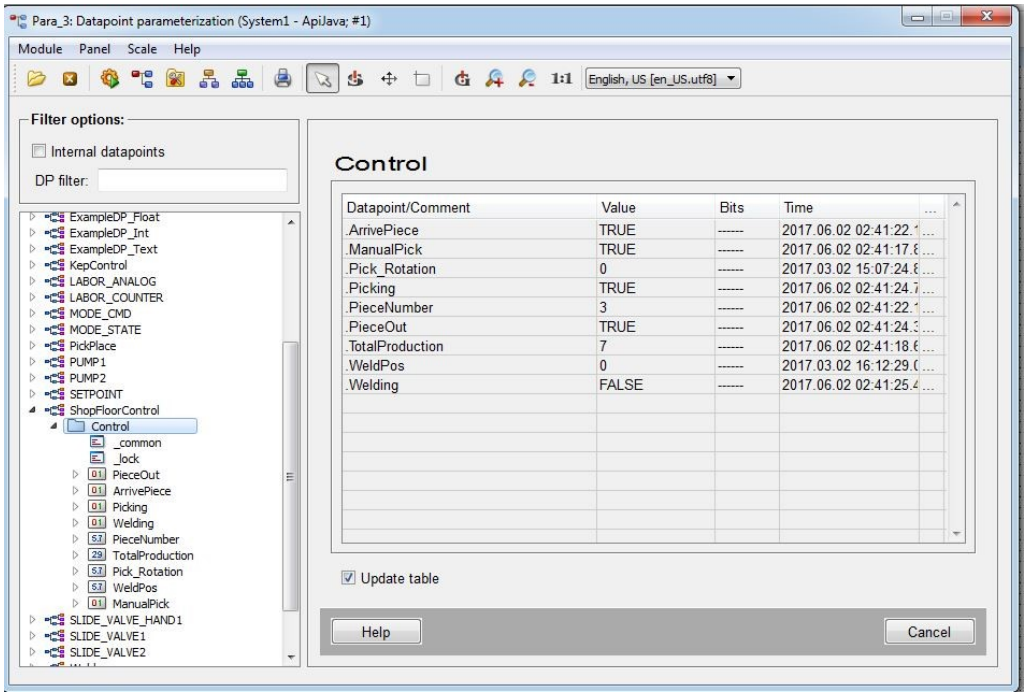

Figura 6.3: *DataPoint* e respetivos Nós

Como se pode ver na figura, existe um DP com o nome de *Control*. Este é constituído por vários nós que estão ligados aos valores extraídos do ambiente simulado. Quando existe qualquer alteração nos valores do ambiente simulado os valores no WinCC OA também o são. É desta forma que se consegue uma visualização em tempo real do que está a acontecer no *shop-floor*.

Os nós existentes são:

- ArrivePiece – Valor do sensor existente no fim do primeiro tapete rolante. Este valor é um booleano, ou seja, quando a peça está no fim do tapete pronta a soldar este valor está a *TRUE*, caso contrário *FALSE*;

- PieceOut – Valor que indica se existe uma peça no tapete rolante de saída. Este valor é um booleano e quando alguma peça está no tapete de saída este valor está a *TRUE*, caso contrário *FALSE*;

- Picking – Valor que indica se o robot de *Pick and Place* está a transportar alguma peça, caso afirmativo este nó está com o valor *TRUE*, caso contrário *FALSE*;

- Welding – Valor que indica se o robot de soldadura está a soldar. Caso aconteça o valor é *TRUE*, senão o valor é *FALSE*;

- Pick_Rotation – Valor que indica em graus a rotação do robot de *Pick and Place*;

- WeldPos – Nó que indica a posição do robot de soldadura.

Os restantes nós existem para controlo no WinCC OA e não são influenciados diretamente pelas alterações no *shop-floor*, por exemplo, o TotalProduction é um contador de peças que já foram produzidas.

A linguagem de programação existente em WinCC OA é uma linguagem de scripts e tem o nome de CONTROL (ETM, s.d.-a). Os métodos mais importantes e mais usados para a implementação desta HMI foram:

- getValue() e setValue() – Métodos que permitem ler ou escrever os vários atributos de um determinado elemento gráfico. Por exemplo posição, rotação, cor, entre outras possibilidades;

- dpGet() e dpSet() – Estes dois métodos são usados para ler e escrever os DataPoints e os seus nós;

- dpConnect() – Este método torna-se nesta arquitetura o mais importante, pois permite criar uma ligação a um determinado DP ou nó e ficar à "escuta" de alterações do mesmo. Quando houver qualquer alteração é executado um outro método específico. Por exemplo, quando o robot de solda está a executar a sua tarefa existe alteração na sua posição e no DP Control no nó WeldPos vai também existir mudanças e visto que esse nó está a ser escutado vai ser executado um método que vai movimentar na HMI o respetivo robot.

Para o DataPoint ficar ligado e à escuta dos valores do ambiente simulado é necessário configurar esta ligação. Devido a algumas restrições quer da parte do WinCC OA quer do V-REP foi necessário usar um terceiro software para criar o elo de ligação entre esse dois. Algumas destas restrições são: a compatibilidade do WinCC OA apenas com Windows versão Profissional o que obrigou a utilização de uma máquina virtual; a linguagem de programação usada em V-REP é Lua (Roberto Ierusalimschy, Waldemar Celes & Luiz Henrique de Figueiredo, s.d.) e esta não permite ligar diretamente estes dois softwares.

De forma a resolver este problema, utilizou-se o KepServer (Kepware, s.d.) que funciona como um servidor, que permite a ligação e transmissão de dados de vários elementos. E assim, após configurar um cliente no WinCC OA existe uma ligação indireta entre este e o V-REP.

6.3 Interação entre WinCC OA e o V-REP

Como referido no subcapítulo anterior, foi usado o KepServer para ser o elo de ligação entre WinCC OA e o V-REP. Para cada valor existente e necessário no V-REP foi criado uma *Tag* no KepServer como se pode ver na figura 6.4.

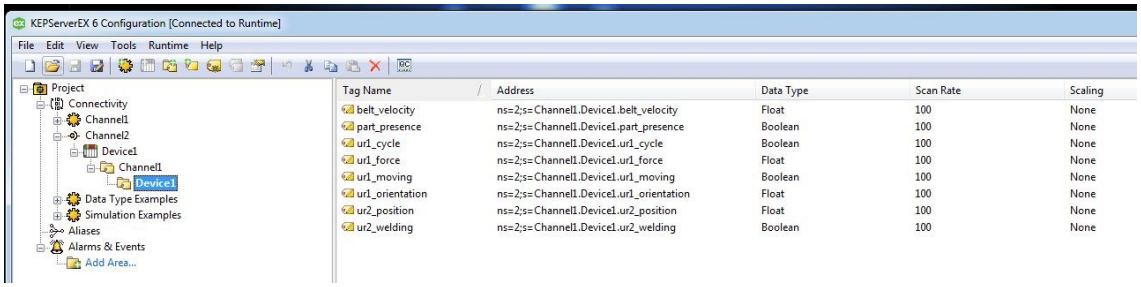

Figura 6.4: Lista de *TAG's* criadas no KepServer

Temos oito Tags definidas como a velocidade do tapete, a força aplicada pelo robot de *Pick and Place* ou a orientação do mesmo. Cada uma destas Tags está ligada a cada um dos nós definidos no WinCC OA, anteriormente explicados. O próximo passo necessário foi a configuração no WinCC OA.

Primeiro é necessário a definição do servidor de onde se pretendia obter informações. Esta definição está ilustrada na figura 6.5.

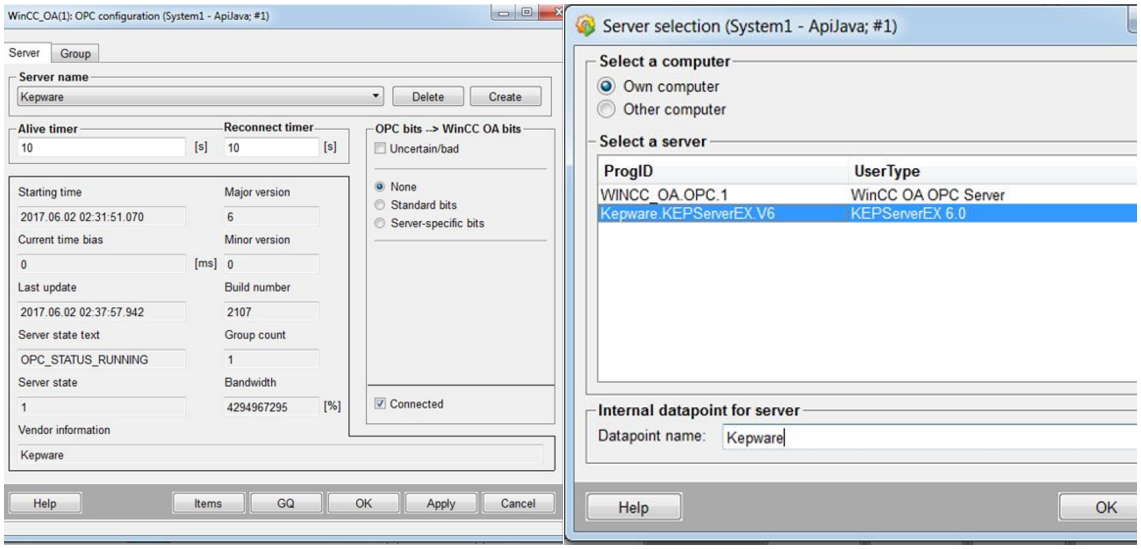

Figura 6.5: Configuração do Servidor no WinCC OA

Como se pode ver no lado esquerdo da imagem 6.5 é possível verificar várias informações referentes ao servidor, como se está a correr ou não, o *timetamp* da última atualização, entre outros.

Além desta configuração é necessário especificar no ficheiro de configuração do projeto do WinCC OA que existe um servidor definido. O ficheiro de configuração está representado na figura 6.6.

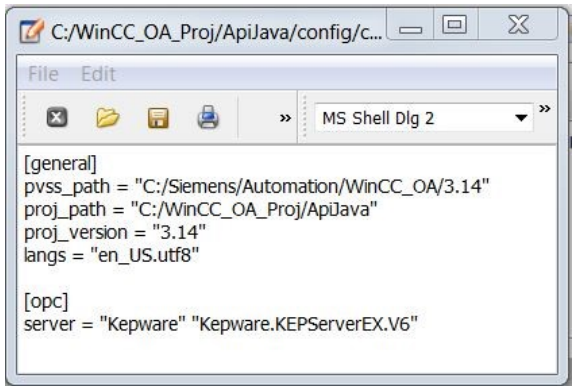

Figura 6.6: Ficheiro de configuração do projeto do HMI

Como se pode verificar, é necessário dizer que tipo de servidor foi definido, neste caso OPC e o nome do mesmo, nas configurações do projeto. Para o cliente ficar funcional é necessário a criação de um *Manager* do tipo *OPC UA Client*, para ser possível ler os dados do servidor. A lista de *Managers* está representada na figura 6.7.

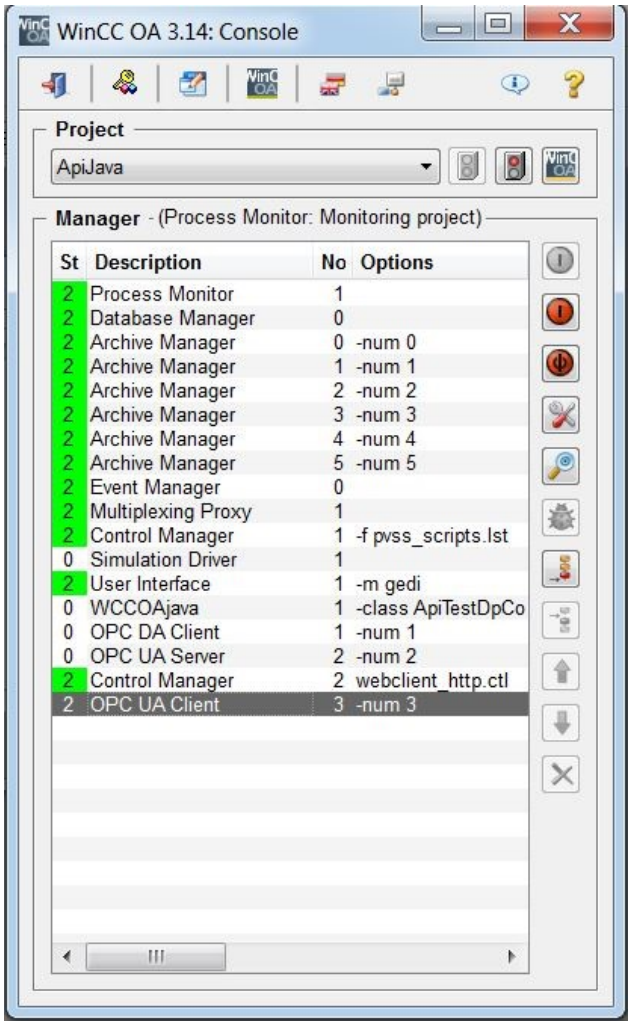

Figura 6.7: Lista de *Managers* presentes no projeto do HMI

Na figura pode-se ver um número grande de *Managers*, sendo que a maioria é automaticamente criada com cada projeto, e não vão ser descritos aqui. O *Manager* criado está selecionado na imagem. Outro Manager criado foi um Control Manager com o número 2 e com a opção *webclient_http.ctl*, sendo que este é usado para disponibilizar o HMI e os dados no browser ou na aplicação do *smarphone*.

Com a definição do servidor, a alteração do ficheiro de configuração e a criação do Manager correspondente termina-se as configurações necessárias para ser possível aceder aos dados do KepServer.

Para cada nó que se pretende ligar a uma *Tag* no KepServer, é necessário adicionar uma configuração ao mesmo. Esta configuração é um endereço periférico sendo assim possível a atualização dos dados no WinCC quando existe alteração dos dados no KepServer. Esta configuração está descriminada na figura 6.8.

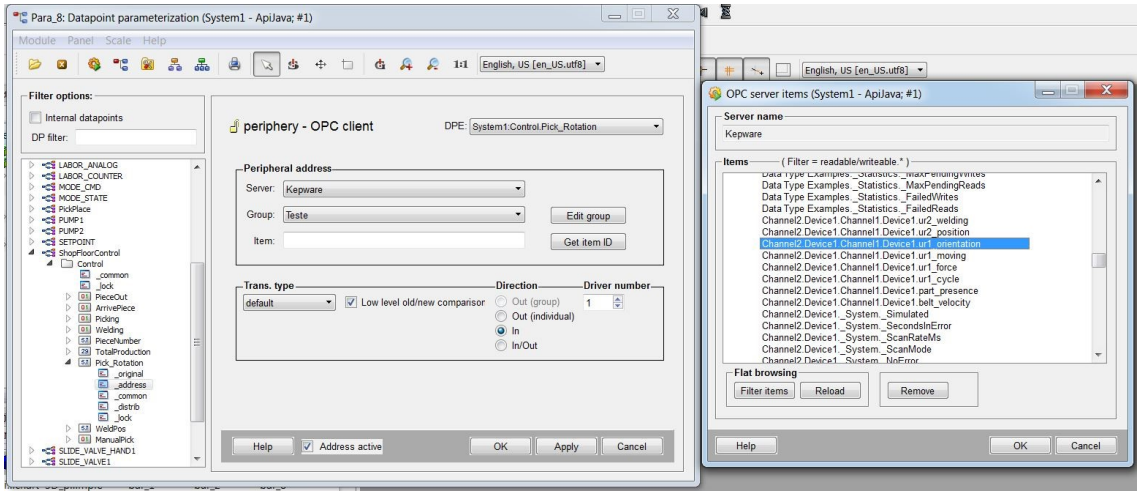

Figura 6.8: Configuração do Valor Periférico de um Nó

Como se pode ver na figura, para configurar o endereço periférico é necessário indicar o Servidor de onde se pretende ler e o Item pretendido. A escolha do Item é feita a partir da lista de elementos fornecidos pelo servidor. Esta lista pode ser vista no lado direito da figura. Assim todas as configurações necessárias estão realizadas.

A imagem 6.9 apresenta o sistema completo constituído pelo *shop-floor*, o middleware e as ferramentas presentes no alto nível e as tecnologias usadas em cada camada.

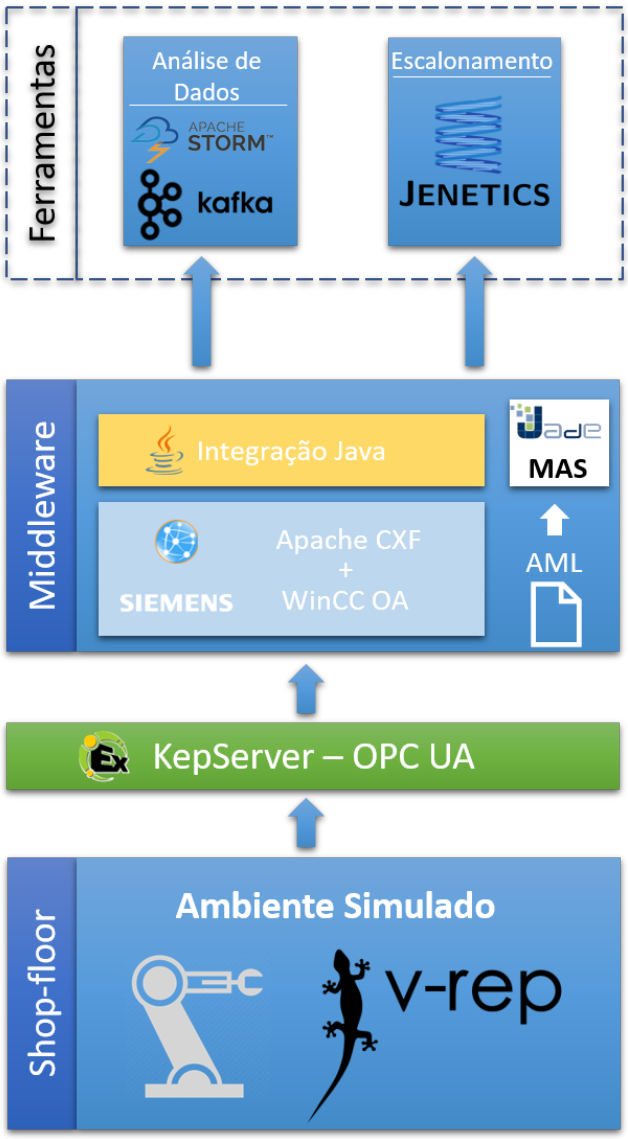

Figura 6.9: Arquitetura completa mais tecnologias

Nas secções anteriores foram apresentadas as interações entre Middleware e o *Shop-floor*. Nos próximos subcapítulos vão ser descritas as interações entre as ferramentas software e o restante sistema.

6.4 Interação com a ferramenta de Análise de Dados

Como referido, no grupo RICS foram desenvolvidas duas aplicações software para o H2020 PERFoRM que irão ser utilizadas para validar os resultados do trabalho desenvolvido.

Uma destas ferramentas consiste numa framework de aquisição e análise de dados. Esta aplicação é apresentada e descrita por Peres (Peres et al., 2017) como uma framework que implementa um sistema de aquisição e análise de dados de uma forma altamente

flexível e distribuída. Este sistema vai oferecer suporte nas tomadas de decisão em tempo real e poderá ativar mecanismos de ajuste automático do sistema. Estes ajustes são proativos, isto é, quando o sistema deteta que poderá haver uma falha vão ser executadas correções no sistema para a prevenir atempadamente, reduzindo assim os impactos de falhas no sistema.

A implementação desta ferramenta foi feita seguindo uma estrutura por camadas, permitindo assim que a complexidade geral da arquitetura seja reduzida. Existem cinco camadas diferentes na arquitetura, são elas: *Shop-floor*, *Data Aquisition*, *Data Queue*, *Data Analysis* e *Data Visualization*.

Como descrito em (Peres et al., 2017) a camada de *Data Aquisition* foi implementada usando uma solução baseada em *Multi-Agent System* (MAS). Foram implementados dois tipos de agentes, os *Component Monitoring Agent* (CMA) que são responsáveis por abstrair componentes individuais existentes no *shop-floor*, recolher todos os dados relevantes e executar um pré-processamento dos mesmos. Estes CMA podem ser associados aos *Subsystem Monitoring Agent* (SMA), onde transmitem os dados recolhidos e pré-processados para serem extraídos num nível de abstração maior. O funcionamento dos SMA é como os CMA mas são responsáveis por abstrair um subsistema.

Para as camadas de mais alto nível foram usadas as tecnologias *Apache Kafka* e *Apache Storm* e a descrição das mesmas podem ser encontradas em (Peres et al., 2017).

A figura 6.10 mostra de forma sucinta as várias tecnologias presentes nesta ferramenta e o fluxo que os dados tomam a partir do WinCC OA.

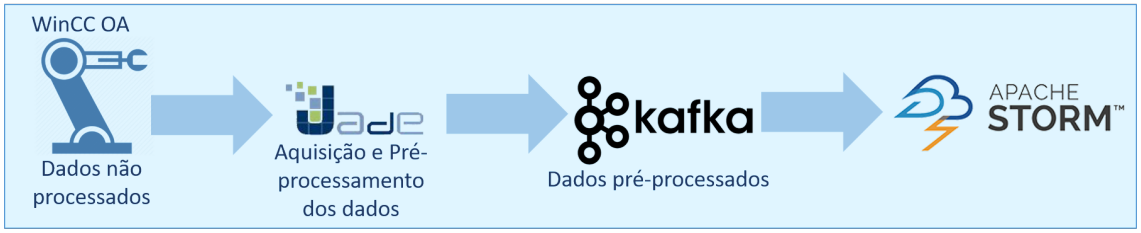

Figura 6.10: Fluxo de Dados desde o WinCC OA

Por cada componente existente é criado um CMA e estes agentes vão implementar a Interface Standard. Os dados são extraídos do WinCC OA por estes agentes que subscrevem os valores associados, ou seja, qualquer alteração que os valores sofram, os agentes vão ser atualizados. Estes valores sofrem um pré-processamento nos próprios agentes e são transmitidos para o Kafka e seguidamente para o Storm.

A topologia do sistema está guardada no ficheiro aml e representada no WinCC. A ferramenta ao ser iniciada, vai ler a topologia do sistema e criar os agentes necessários para abstrair os vários componentes existentes no sistema. Um exemplo disso está explícito na figura 6.11, de acordo com uma determinada topologia (ficheiro aml representado no lado direito da figura) são criados os agentes que se podem ver na janela ao centro da mesma (MA_ConveyorBelt, MA_UR5_1 e MA_UR5_2).

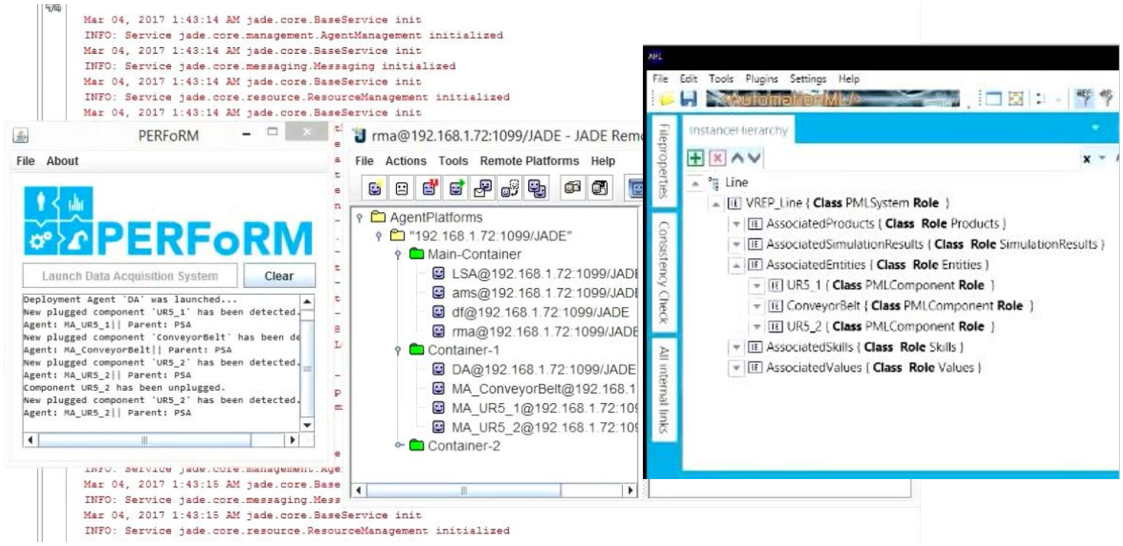

Figura 6.11: Criação de Agentes segundo o ficheiro aml

Por cada máquina existe um *Deployment Agent* (DA) que é responsável por manter a topologia atualizada, ou seja, quando são inseridos ou removidos componentes do sistema criam-se ou destroem-se agentes mantendo sempre tudo coerente.

Na figura 6.12 é possível ver a ferramenta em ação, onde de um lado se encontram os dados extraídos do WinCC OA e do outro o resultado da framework.

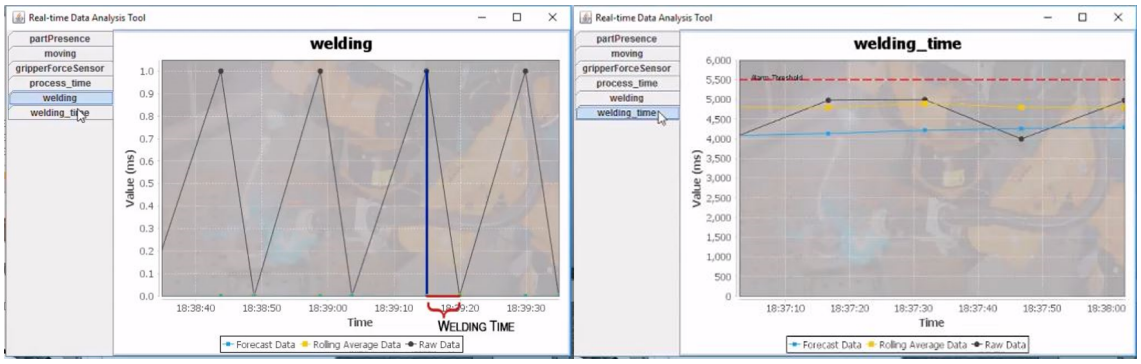

Figura 6.12: Resultados da Ferramenta de Análise de Dados

Do lado esquerdo da figura, pode ver-se a extração de dados (provenientes do WinCC OA) ao longo do tempo consoante as suas alterações. Na imagem está representado o movimento completo do robot de soldar, desde que parte da posição inicial, solda e volta à sua posição. Do lado direito da figura encontra-se o tempo de soldadura (dados não processados) assim como os resultados da ferramenta de análise de dados.

Neste subcapítulo verificou-se com sucesso a interligação da implementação apresentada no documento com uma ferramenta desenvolvida por terceiros. Na próxima secção o trabalho desenvolvido vai ser testado usando uma segunda aplicação software.

6.5 Interação com o Escalonador

O Escalonador é uma ferramenta que serve para escalonar as várias operações segundo um determinado requisito, para um certo intervalo de tempo. Este escalonamento é feito com base no estado do sistema, e das operações quer de produção quer de manutenção que se pretende que sejam executadas.

O escalonador vai gerar vários resultados, todos eles válidos, e cabe ao utilizador final escolher o que mais se adequa a uma determinada situação. Existem diferentes requisitos para gerar os escalonamentos, como por exemplo começar o processo de produção o mais depressa possível ou ser o processo mais eficiente.

A imagem 6.13 mostra a interface gráfica existente no escalonador.

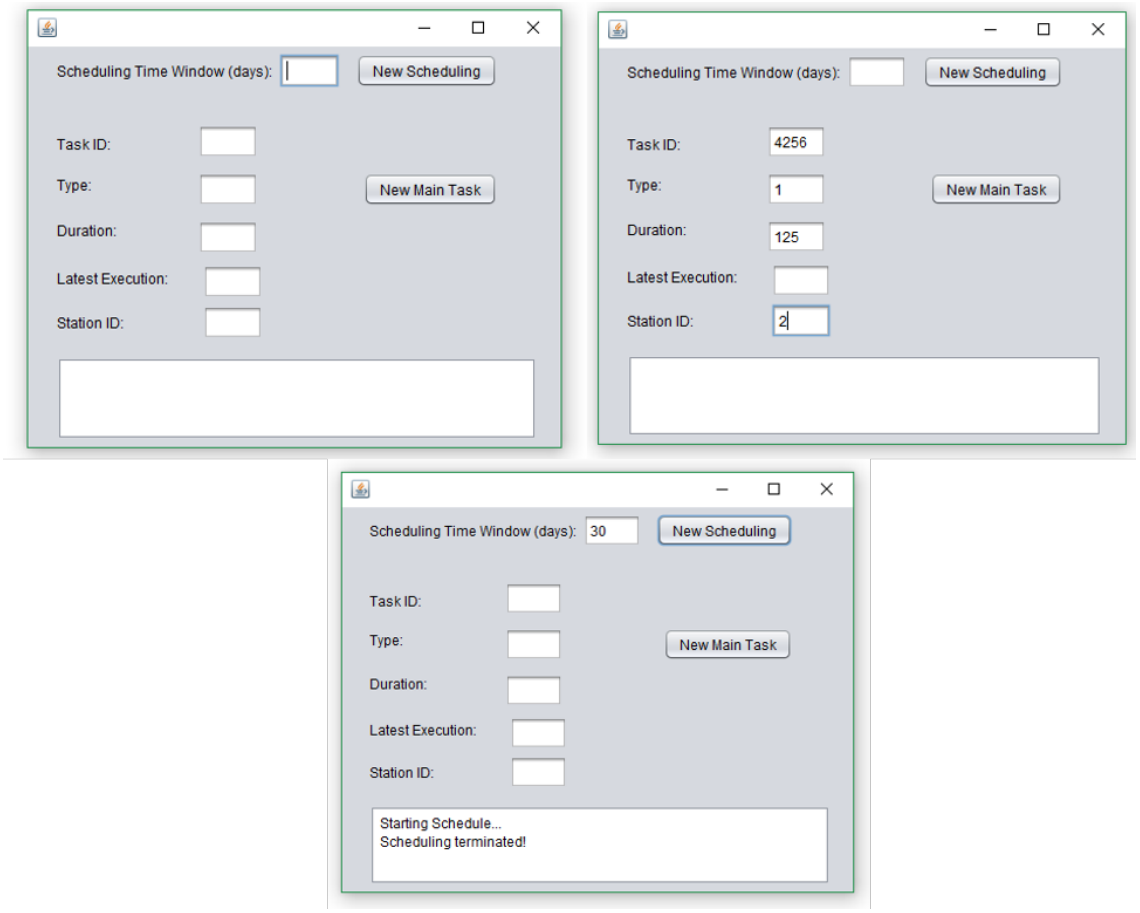

Figura 6.13: Interface Gráfica do Escalonador

Na figura é possível ver a criação quer, de uma nova tarefa de manutenção, como a geração de um escalonamento considerando 30 dias de produção.

O Escalonador precisa de algumas informações para conseguir gerar os seus escalonamentos. As informações necessárias são a lista de operações que é necessário considerar (esta informação é da responsabilidade do cliente quando faz o pedido, como se pode verificar pela Interface definida na subsecção 5.2.1) e a topologia do sistema, para selecionar

as máquinas apropriadas.

A topologia está definida num ficheiro aml no Middleware, e assim para obter essa informação, o Escalonador vai fazer um pedido ao Middleware usando a mesma Interface referida em cima. Um exemplo do resultado deste pedido está representado na figura 6.14.

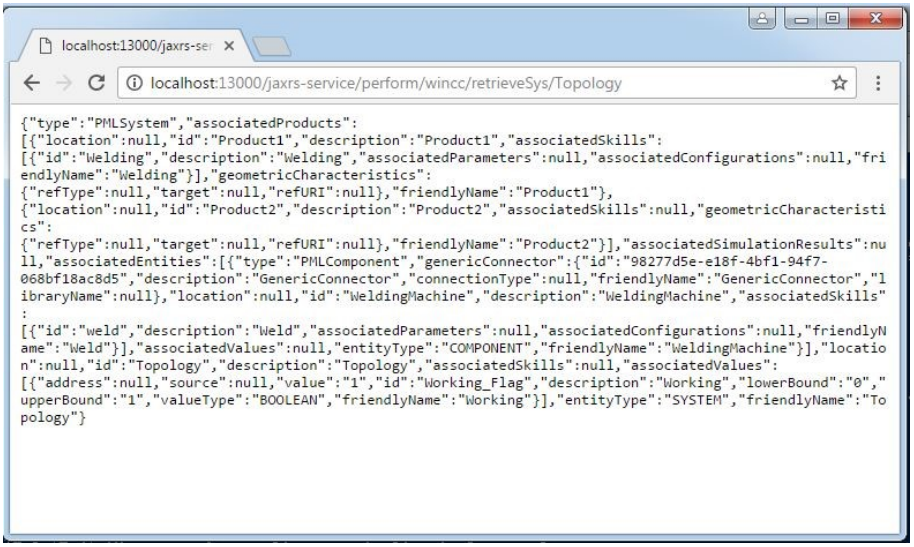

Figura 6.14: Resposta do Middleware ao Escalonador

Como referido anteriormente, a troca de dados é através de JSON e assim a *string* resultante do pedido da Topologia ao Middleware está descriminada na figura 6.14. Como se pode verificar a topologia definida contém dois produtos e um componente definidos.

Após a geração do Escalonamento, este resultado é enviado ao Middleware que irá entregar ao cliente. Um exemplo de um resultado do Escalonador recebido pelo middleware pode ser visto na figura 6.15.

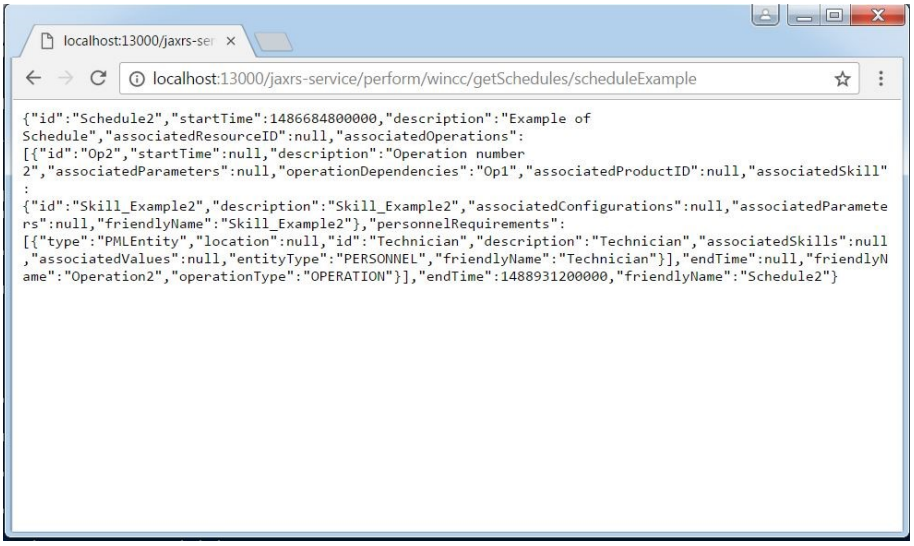

Figura 6.15: Resposta do Escalonador ao Middleware

Mais uma vez, como a troca de dados é JSON, pode facilmente ver-se a *string* transmitida pelo Escalonador e recebida pelo Middleware.

Como foi possível verificar nesta última secção a integração com o escalonador foi conseguida, validando mais uma vez as Interfaces Standard e o Modelo de Dados apresentados neste documento.

O propósito da implementação proposta era mostrar uma possível solução para integrar de maneira transparente os vários elementos heterogéneos de um sistema seguindo o conceito da Indústria 4.0 e como se pode verificar por este último capítulo a solução proposta conseguiu integrar vários elementos e várias tecnologias com sucesso.

Apesar de não ter sido testada em ambiente real, para garantir com certezas os resultados, a arquitetura portou-se como o expectável, e assim pode-se concluir que a implementação proposta é realmente uma possível solução de integração dentro do conceito da Indústria 4.0.

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Conclusões

No contexto da Indústria 4.0, a interoperabilidade é um ponto fulcral. Assim sendo, neste documento foi proposto um Modelo de Dados Comum genérico capaz de abstrair os diferentes elementos heterogéneos presentes num Sistema de Produção e Interfaces Standards para expor as funcionalidades dos mesmos, garantindo assim uma integração transparente.

O Modelo de Dados Comum é genérico o suficiente para abstrair elementos presentes em diferentes níveis arquitetónicos, isto é, abstrai elementos existentes quer no alto nível (aplicações software) quer no baixo nível (hardware). Já as Interfaces Standard são capazes de expor todas as funcionalidades dos elementos do sistema na forma de serviços, garantindo assim uma maior interoperabilidade e plugabilidade do sistema.

Com o trabalho apresentado e os testes descritos neste documento foi possível concluir que uma abordagem onde a descrição de um Sistema de Produção através de um Modelo de Dados Comum genérico e de Interfaces Standard é válida e adequada num contexto da Indústria 4.0 para a integração de elementos heterogéneos.

Apesar dos testes realizados no capítulo 5 terem sido feitos com base num *shop-floor* simulado, foi possível verificar que a abordagem e as tecnologias escolhidas são aplicáveis e garantem a interoperabilidade dos vários elementos heterogéneos através de um Modelo de Dados Comum e de Interfaces Standard.

O trabalho desenvolvido está inserido no projeto H2020 PERFoRM. Neste âmbito, foi realizado um workshop em Bragança em colaboração com o Instituto Politécnico de Bragança (IPB), o grupo Loccioni e a University of Applied Sciences Hochschule Emden/Leer (HSEL), para apresentar o trabalho conjunto incluindo uma demonstração do mesmo.

Este workshop levou à submissão de uma publicação científica com o nome *Integration*

and Deployment of a Distributed and Pluggable Industrial Architecture for the PERFoRM Project (Angione et al., 2017) para a *27th International Conference on Flexible Automation and Intelligent Manufacturing* (FAIM 2016). Este artigo foi aceite e apresentado em Junho de 2017.

Está ainda a ser preparado outro artigo científico no âmbito da integração do Modelo de Dados Comum desenvolvido com a ferramenta de Escalonamento.

7.2 Trabalho Futuro

No que diz respeito à realização de trabalhos futuros, existe a necessidade de execução de testes em ambientes reais. Como o trabalho presente neste documento está inserido no H2020 PERFoRM o próximo passo será instanciar o modelo de dados nos quatro demonstradores que estão em estado de desenvolvimento. Estes demonstradores abrangem várias áreas da Indústria Europeia como Eletrodomésticos (Whirlpool), Aeroespacial (GKN), Micro veículos elétricos (IFEVS) e produção de compressores de grandes dimensões (Siemens).

Outra abordagem necessária é o estudo do impacto que as tecnologias escolhidas têm num sistema real de produção, ou seja, verificar vários parâmetros como por exemplo tempos de comunicação e atrasos.

Como referido anteriormente, o modelo de dados muitas vezes necessita de ser atualizado consoante as necessidades que vão surgindo, e assim sendo o modelo de dados apresentado pode possivelmente ser alterado à medida que novos requisitos forem surgindo no decorrer do projeto.

Bibliografia

Angione, G., Barbosa, J., Gosewehr, F., Leitão, P., Daniele, M., João, M., ... Wermann, J. (2017). Integration and deployment of an industrial architecture for the PERFoRM project. *27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017*.

Apache Software Foundation. (s.d.). Apache Camel. Obtido 25 maio 2017, de http://camel.apache.org/

Apache Software Foundation. (2017). Apache CXF. Obtido 9 junho 2017, de http://cxf.apache.org/

Arnold, K. (1999). The Jini architecture: dynamic services in a flexible network. Em *Design automation conference, 1999. proceedings. 36th* (pp. 157–162). IEEE. doi:10.1109/DAC.1999.781302

Atzori, L., Iera, A. & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787–2805. doi:10.1016/j.comnet.2010.05.010

Babiceanu, R. F. R. F. & Chen, F. F. (2006). Development and applications of holonic manufacturing systems: a survey. *Journal of Intelligent Manufacturing*, *17*(1), 111–131. doi:10.1007/s10845-005-5516-y

Barata, J., Santana, P. & Onori, M. (2006). Evolvable assembly systems: a development roadmap.

Barbosa, J. (2015). Self-organized and evolvable holonic architecture for manufacturing control. *PHD Theses*.

Bobek, A., Zeeb, E., Bohn, H., Golatowski, F. & Timmermann, D. (2008). Devices Profile for Web Services (DPWS). doi:10.1109/INDIN.2008.4618210

Bohn, H., Bobek, A. & Golatowski, F. (2006). SIRENA-Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains. *... , International Conference on ...*

Buxmann, P., Hess, T. & Ruggaber, R. (2009). Internet of services. *Business & Information Systems*.

Chen, F., Deng, P., Wan, J., Zhang, D., Vasilakos, A. V. & Rong, X. (2015). Data Mining for the Internet of Things: Literature Review and Challenges. *International Journal of Distributed Sensor Networks*, *11*(8), 431047. doi:10.1155/2015/431047

Chen, M., Mao, S. & Liu, Y. (2014). Big data: A survey. Em *Mobile networks and applications* (Vol. 19, *2*, pp. 171–209). Springer US. doi:10.1007/s11036-013-0489-0

Christensen, E., Curbera, F., Meredith, G. & Weerawarana, S. (2001). Web Service Definition Language (WSDL). Obtido 18 maio 2017, de https://www.w3.org/TR/wsdl

ElMaraghy, H. (2005). Flexible and reconfigurable manufacturing systems paradigms. *International journal of flexible manufacturing systems*.

EPoSS. (2008). Internet of Things in 2020 A ROADMAP FOR THE FUTURE. *European Technology Platform on Smart Systems Integration*, 1–27.

ETM. (s.d.-a). Control script Language.

ETM. (s.d.-b). WinCC Open Architecture. Obtido 25 maio 2017, de http://www.etm.at/index%7B%5C_%7De.asp?id=2

FAIM. (2016). FAIM 2017. Obtido 20 junho 2017, de http://www.faim2017.org/

Faltinski, S., Niggemann, O., Moriz, N. & Mankowski, A. (2012). AutomationML: From data exchange to system planning and simulation. Em *2012 ieee international conference on industrial technology, icit 2012, proceedings* (pp. 378–383). IEEE. doi:10.1109/ICIT.2012.6209967

Fei Tao, Ying Cheng, Li Da Xu, Lin Zhang & Bo Hu Li. (2014). CCIoT-CMfg: Cloud Computing and Internet of Things-Based Cloud Manufacturing Service System. *IEEE Transactions on Industrial Informatics*, *10*(2), 1435–1442. doi:10.1109/TII.2014.2306383

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* (tese de doutoramento, UNIVERSITY OF CALIFORNIA, IRVINE).

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A. & Lafon, Y. (2007). SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Obtido 18 maio 2017, de https://www.w3.org/TR/soap12/

Hannelius, T., Salmenperä, M. & Kuikka, S. (2008). Roadmap to adopting OPC UA. Em *Ieee international conference on industrial informatics (indin)* (pp. 756–761). IEEE. doi:10.1109/INDIN.2008.4618203

Hermann, M., Pentek, T. & Otto, B. (2016). Design principles for industrie 4.0 scenarios. *Proceedings of the Annual Hawaii International Conference on System Sciences*, *2016-March*(01), 3928–3937. doi:10.1109/HICSS.2016.488. arXiv: arXiv:1011.1669v3

IBM Corporation. (s.d.). IBM Integration Bus. Obtido 25 maio 2017, de http://www-03.ibm.com/software/products/pt/ibm-integration-bus

IBM Corporation. (2016). IBM Integration Bus Manufacturing Pack. Obtido 25 maio 2017, de https://www.ibm.com/support/knowledgecenter/SSMKHH%7B%5C_%7D9.0.0/com.ibm.manufacturing.doc/ma00010.htm

Ing Reiner Anderl. (2014). Industrie 4 . 0 - Advanced Engineering of Smart Products and Smart Production. *International Seminar on High Technology*.

INTERNATIONAL ELECTROTECHNICAL COMMISSION. (2014). IEC 62714-1:2014.

INTERNATIONAL ELECTROTECHNICAL COMMISSION. (2016). IEC 62424:2016.

Jahromi, M. H. M. A. & Tavakkoli-Moghaddam, R. (2012). A novel 0-1 linear integer programming model for dynamic machine-tool selection and operation allocation

in a flexible manufacturing system. *Journal of Manufacturing Systems*, *31*(2), 224–231. doi:10.1016/j.jmsy.2011.07.008

Jovanovi, S. (2015). Flexible Manufacturing Systems and Quantitative Anlysis of Flexible Manufacturing Systems. *International Journal of Computer Applications*, *132*(1), 6–15.

Kagermann, H., Wahlster, W. & Helbig, J. (2013). *Recommendations for implementing the strategic initiative INDUSTRIE 4.0*.

Kepware. (s.d.). KEPServerEX Connectivity Platform. Obtido 9 junho 2017, de https://www.kepware.com/en-us/products/kepserverex/

Koestler & Arthur. (1968). *The ghost in the machine.* Macmillan.

Lee, E. A. (2008). Cyber Physical Systems: Design Challenges. *Proc. of 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'08)*, 363–369. doi:10.1109/ISORC.2008.25

Leitão, P. (2009). Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, *22*(7), 979–991. doi:10.1016/j.engappai.2008.09.005

Leitner, S.-H. & Mahnke, W. (2006). OPC UA – Service-oriented Architecture for Industrial Applications. *Softwaretechnik-Trends*, *26*(4), 1–6.

Lepuschitz, W., Lobato-Jimenez, A., Axinia, E. & Merdan, M. (2015). A survey on standards and ontologies for process automation. Em *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)* (Vol. 9266, pp. 22–32). Springer, Cham. doi:10.1007/978-3-319-22867-9_3

Lucke, D., Constantinescu, C. & Westkämper, E. (2008). Smart Factory - A Step towards the Next Generation of Manufacturing. Em *Manufacturing systems and technologies for the new frontier* (pp. 115–118). London: Springer London. doi:10.1007/978-1-84800-267-8_23

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J. & Ghalsasi, A. (2011). Cloud computing - The business perspective. *Decision Support Systems*, *51*(1), 176–189. doi:10.1016/j.dss.2010.12.006

Martin. (2017). Industry 4.0: Definition, Design Principles, Challenges, and the Future. Obtido 18 maio 2017, de https://www.cleverism.com/industry-4-0/

Miller, B. A., Nixon, T., Tai, C. & Wood, M. D. (2001). Home networking with Universal Plug and Play. *IEEE Communications Magazine*, *39*(12), 104–109. doi:10.1109/35.968819

MTConnect Institure. (s.d.). MTConnect. Obtido 26 maio 2017, de http://www.mtconnect.org/

Onori, M. & Barata, J. (2009). Evolvable production systems: mechatronic production equipment with process-based distributed control.

Orio, G. D., Rocha, A., Ribeiro, L. D. & Barata, J. (2015). The PRIME Semantic Language: Plug and Produce in Standard-based Manufacturing Production Systems. *Flexible Automation and Intelligent Manufacturing (FAIM 2015)*, (July 2016), 8.

Peres, R. S., Parreira-Rocha, M., Rocha, A. D., Barbosa, J., Leitão, P. & Barata, J. (2016). Selection of a data exchange format for industry 4.0 manufacturing systems. Em *Iecon 2016 - 42nd annual conference of the ieee industrial electronics society* (pp. 5723–5728). IEEE. doi:10.1109/IECON.2016.7793750

Peres, R. S., Rocha, A. D. & Barata, J. (2017). Dynamic Simulation for MAS-Based Data Acquisition and Pre-processing in Manufacturing Using V-REP. Em *Doctoral conference on computing*, (pp. 125–134). doi:10.1007/978-3-319-56077-9_11

PERFoRM Project. (2016a). Deliverable 1.2 - Requirements for Innovative Production System Functional requirement analysis and definition of strategic objectives and KPIs, 93.

PERFoRM Project. (2016b). *Deliverable 2.2 - Definition of the System Architecture*.

PERFoRM Project. (2016c). PERFoRM - Production harmonizEd Reconfiguration of Flexible Robots and Machinery. Obtido 31 maio 2017, de http://www.horizon2020-perform.eu/

PERFoRM Project. (2017). *Deliverable 2.3 - Specification of the Generic Interfaces for Machinery, Control Systems and Data Backbone*.

Picard, A., Anderl, R. & Schützer, K. (2013). Linked Product and Process Monitoring in Smart Factories Based on Federative Factory Data Management. *ASME 2013*.

Qiu, M., Xue, C., Shao, Z., Zhuge, Q., Liu, M. & Sha, E. H. .-.-M. (2006). Efficent Algorithm of Energy Minimization for Heterogeneous Wireless Sensor Network. (pp. 25–34). Springer, Berlin, Heidelberg. doi:10.1007/11802167_5

Rahul. (2017). IoT applications spanning across industries. Obtido 29 maio 2017, de https://www.ibm.com/blogs/internet-of-things/iot-applications-industries/

Red Hat. (s.d.). Red Hat JBoss Fuse. Obtido 25 maio 2017, de https://www.redhat.com/en/technolog-%20ies/jboss-middleware

Ribeiro, L., Barata, J. & Pimentão, J. (2011). Where evolvable production systems meet complexity science. *Assembly and Manufacturing (...*

Ribeiro, L. & Barata, J. (2011). Re-thinking diagnosis for future automation systems: An analysis of current diagnostic practices and their applicability in emerging IT based production paradigms. Elsevier B.V. doi:10.1016/j.compind.2011.03.001

Roberto Ierusalimschy, Waldemar Celes & Luiz Henrique de Figueiredo. (s.d.). The Programming Language Lua. Obtido 19 agosto 2017, de https://www.lua.org/

Rocha, A., Di Orio, G., Barata, J., Antzoulatos, N., Castro, E., Scrimieri, D., ... Ribeiro, L. (2014). An agent based framework to support plug and produce. *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, 504–510. doi:10.1109/INDIN.2014.6945565

SAP. (s.d.). Manufacturing Intelligence and Integration. Obtido 25 maio 2017, de https://www.sap.com/products/manufacturing-intelligence-integration.html%7B%5C#%7D

Schleipen, M., Drath, R. & Sauer, O. (2008). The system-independent data exchange format CAEX for supporting an automatic configuration of a production monitoring and control system. Em *Ieee international symposium on industrial electronics* (pp. 1786–1791). IEEE. doi:10.1109/ISIE.2008.4676932

Schuh, G., Potente, T., Wesch-Potente, C., Weber, A. R. & Prote, J.-P. (2014). Collaboration Mechanisms to Increase Productivity in the Context of Industrie 4.0. *Procedia CIRP*, *19*, 51–56. doi:10.1016/j.procir.2014.05.016

Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S. & Xu, X. (2014). Web services composition: A decade's overview. *Information Sciences*, *280*, 218–238. doi:10.1016/j.ins.2014.04.054. arXiv: arXiv:1011.1669v3

Spyns, P., Meersman, R. & Jarrar, M. (2002). Data modelling versus ontology engineering. *ACM SIGMOD Record*, *31*(4), 12. doi:10.1145/637411.637413

TongKe, F. (2013). Smart agriculture based on cloud computing and IOT. *Journal of Convergence Information*.

Ueda, K. (1992). A concept for bionic manufacturing systems based on DNA-type information. *Proceedings of the IFIP TC5/WG5. 3 Eight International ...*

Ueda, K., Hatono, I., Fujii, N. & Vaario, J. (2000). Reinforcement learning approaches to biological manufacturing systems. *CIRP Annals-Manufacturing ...*

Wan, J., Cai, H. & Zhou, K. (2015). Industrie 4.0: Enabling technologies. Em *Proceedings of 2015 international conference on intelligent computing and internet of things* (pp. 135–140). IEEE. doi:10.1109/ICAIOT.2015.7111555

Wang, S., Wan, J., Li, D. & Zhang, C. (2016). Implementing Smart Factory of Industrie 4.0: An Outlook. *International Journal of Distributed Sensor Networks*, *12*(1), 3159805. doi:10.1155/2016/3159805

Wang, X. V. & Xu, X. W. (2013). Robotics and Computer-Integrated Manufacturing An interoperable solution for Cloud manufacturing. *Robotics and Computer Integrated Manufacturing*, *29*(4), 232–247. doi:10.1016/j.rcim.2013.01.005

Wurth Group. (s.d.). Definition of C-Parts. Obtido 9 junho 2017, de https://www.wuerth-industrie.com/

Xu, X. (2012). From cloud computing to cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*, *28*(1), 75–86. doi:10.1016/j.rcim.2011.07.002

Yang, G., Xie, L., Mantysalo, M., Zhou, X., Pang, Z., Xu, L. D., ... Zheng, L.-R. (2014). A Health-IoT Platform Based on the Integration of Intelligent Packaging, Unobtrusive Bio-Sensor, and Intelligent Medicine Box. *IEEE Transactions on Industrial Informatics*, *10*(4), 2180–2191. doi:10.1109/TII.2014.2307795

Zeeb, E., Bobek, A., Bohn, H. & Golatowski, F. (2007). Service-oriented architectures for embedded systems using devices profile for Web services. Em *Proceedings - 21st international conference on advanced information networking and applications workshops/symposia, ainaw'07* (Vol. 2, pp. 956–963). doi:10.1109/AINAW.2007.330